Management Service Group

Management Overview

Accounting

Accounting: Guides

Accounting: Concept Guide

Accounting is a bindery-based service that consists of the functions required for administering and recording accounting events.

Chargeable Services

Disk Storage Charges

Reading the net$acct.dat and net$rec.dat Files Example

Accounting: Functions

Accounting: Structures

Parent Topic:

Management Overview

Accounting: Examples

Reading the net$acct.dat and net$rec.dat Files Example

The example code below allows you to read the net$acct.dat and net$rec.dat files.

Using net$acct.dat and net$rec.dat

```
#include <stdio.h>;
#include <conio.h>
#include <io.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <nwmisc.h>
#include <ntypes.h>

/* Structure of a Charge Record in Net$Acct.Dat */

typedef struct
{
   nuint16   length;
   nuint32   serverID;
   nuint8    timeStamp[6];
   nuint8    recordType;
   nuint8    completionCode;
   nuint16   serviceType;
   nuint32   clientID;
   nuint32   amount;
   nuint16   commentType;
   char      *comment;
}     ChargeRecord;

/* Structure of a Note Record in Net$Acct.Dat */
typedef struct
{
   nuint16   length;
   nuint32   serverID;
   nuint8    timeStamp[6];
   nuint8    recordType;
   nuint8    reserved;
   nuint16   serviceType;
   nuint32   clientID;
   nuint16   commentType;
   pnstr     *comment;
}  NoteRecord;

/* Structure of a Format Record in Net$Rec.Dat */
typedef struct
{
   nuint16   length;
   nuint16   commentType;
   nuint8    fieldCount;
   nuint8    firstDataType[];
} FormatRecord;

/* Structure of a Charge Comment in Net$Acct.Dat */
/* Connect Time Charge, Disk Space Charge        */
typedef struct
{
   nuint32   unit1;
   nuint32   unit2;
   nuint16   bytesRead[3];   /* only used in Connect Time Charge Commen
   nuint16   bytesWritten[3];/* only used in Connect Time Charge Commen
}  ChargeComment;

/* Structure of a Node Comment in Net$Acct.Dat */
/* Login, Logout, Account locked notes         */
typedef struct
{
   nuint32   net;
   nuint32   nodeHi;
   nuint16   nodeLo;
} NodeComment;


/* Prototype for the function that reads Net$Rec.Dat and returns */
/* the format string                                             */
nuint GetFormatString
(
   pnstr     formatString,
   pnuint8   fieldCount,
   pnstr     dataType,
   nuint16   commentType
);


/* Prototype for the function that prints a Connect Time */
/* charge record to screen                               */
int PrintConnectTimeRecord
(
   pnstr           format,
   ChargeComment   *record,
   double          bytesRead,
   double          bytesWritten
```

```
);


/* Prototype for the function that prints a charge record to screen */
int   PrintChargeRecord
(
   pnstr          format,
   ChargeRecord   *record
);


/* Prototype for the function that prints a note record to screen */
int   PrintNoteRecord
(
   pnstr        format,
   NoteRecord   *note
);


/* !!!!!!  Function main  !!!!!! */

void main (void)
{
   int    ccode;        /* Variable to receive return codes
   FILE   *acct;        /* Pointer to the stream associated with Net$Ac
   nstr   buffer[255];  /* Buffer to read the record from Net$Acct.Dat

   ChargeRecord    *charge;
   ChargeComment   *chargeUnit;
   NoteRecord      *note;
   NodeComment     *node;
   nuint8          fieldCount;
   nstr            dataType[15];
   nstr            format[255];


   clrscr();

   printf("Novell sample program to print records from the Net$Acct.Dat
           file\n\n");

   /* Open the stream for Net$Acct.Dat */
   if (access("NET$ACCT.DAT", 04))
         acct = fopen("NET$ACCT.DAT", "rb");/*  File is in default dire
   else
      acct = fopen("SYS:PUBLIC\NET$ACCT.DAT", "rb");
      /* File is not in default dir.  */
   if (NULL == acct)
   {
      printf("\tUnable to open Net$Acct.Dat\n");
      printf("\tExiting program\n\n");
      exit(1);
```

```
}

/*  Overlay the accounting structures on top of buffer  */
charge = (ChargeRecord *)buffer;
chargeUnit = (ChargeComment *)&charge->comment;

note  = (NoteRecord * ) buffer;
node = (NodeComment *)&note->comment;

do
{

   /* Let's read the length of the next record from Net$Acct.Dat  */
   ccode = read(fileno(acct), &charge->length, 2);
   charge->length = NWWordSwap(charge->length);
   printf("Net$Acct.Dat Record length = %d \n", charge->length);

   /* Verify read was successful and length is smaller than buffer s
   if (2 == ccode  &&  253 > charge->length)
   {
      /* Let's read the remainder of the record from Net$Acct.Dat  *
      ccode = read(fileno(acct), &charge->serverID,

      /* Verify read was successful  */
      if (charge->length == ccode)
      {
         switch (charge->recordType)
         {
            case 1 :         /* Charge Record */
               printf("serverID = %ld\n"
                      "recordType = %d\ncompletionCode = %d\n"
                      "serviceType = %d\nclientID = %ld\n"
                      "amount = %ld\ncommentType = %d\n",
                      NWLongSwap(charge->serverID),
                      charge->recordType,
                      charge->completionCode,
                      NWWordSwap(charge->serviceType),
                      NWLongSwap(charge->clientID),
                      NWLongSwap(charge->amount),
                      NWWordSwap(charge->commentType));

               printf("Year: %d\nMonth: %d\nDay: %d\n"
                      "Hour: %d\nMinute: %d\nSecond: %d\n",
                      (int) charge->timeStamp[0],
                      (int) charge->timeStamp[1],
                      (int) charge->timeStamp[2],
                      (int) charge->timeStamp[3],
                      (int) charge->timeStamp[4],
                      (int) charge->timeStamp[5]);

               /* Read the format string from Net$Rec.Dat  */
               ccode = GetFormatString(format,
                       &fieldCount,
                       &dataType[0],
                       NWWordSwap(charge->
                       commentType));

               /* Verify GetFormatString was successful  */
               if (0 == ccode)
               {
                  /* Verify the comment is a Novell charge */
                  /* comment                               */
                  if (1 == NWWordSwap(charge->
                      commentType)||
                      2 == NWWordSwap(charge->
                      commentType))
                  {
                     /* Swap the byte order to Intel format */
                     chargeUnit->unit1==NWLongSwap
                              (charUnit->unit1);
                     chargeUnit->unit2 = NWLongSwap
                              (chargeUnit->unit2);
                  }

                  /* Verify the comment is a Connection Time Comment
                  if (1 == NWWordSwap(charge->commentType))
                  {
                     double bytesRead;
                     double bytesWritten;

                     bytesRead =
                       (NWWordSwap(chargeUnit->bytesRead[0]) * 0xFF0
                       (NWWordSwap(chargeUnit->bytesRead[1]) * 0x00F
                       (NWWordSwap(chargeUnit->bytesRead[2]));
                     bytesWritten =
                       (NWWordSwap(chargeUnit->bytesWritten[0]) * 0x
                       (NWWordSwap(chargeUnit->bytesWritten[1]) * 0x

                     PrintConnectTimeRecord(format,
                           chargeUnit, bytesRead, bytesWritten);
                  }
                  else
                  {
                     PrintChargeRecord(format, charge);
                  }

                  printf("\n\n");
               }
               else
                  printf("An error occured in
                         GetFormatString\n");
               break;

            case 2 :               /* Note Record   */
```

```
                        note = (NoteRecord *)buffer;
                               printf("serverID = %ld\nrecordType = %d\n"
                               "serviceType = %d\nclientID = %ld\n"
                               "commentType = %d\n",
                               NWLongSwap(note->serverID),
                               note->recordType,
                               NWWordSwap(note->serviceType),
                               NWLongSwap(note->clientID),
                               NWWordSwap(note->commentType));

                         printf("Year: %d\nMonth: %d\nDay: %d\n"
                                "Hour: %d\nMinute: %d\nSecond: %d\n",
                                (int) note->timeStamp[0],
                                (int) note->timeStamp[1],
                                (int) note->timeStamp[2],
                                (int) note->timeStamp[3],
                                (int) note->timeStamp[4],
                                (int) note->timeStamp[5]);

                          /* Read the format string from Net$Rec.Dat   */
                          ccode = GetFormatString(format,
                                 &fieldCount,
                                 &dataType[0],
                                  NWWordSwap(note->
                                  commentType));

                           /* Verify GetFormatString was successful  */
                           if (0 == ccode)
                           {
                              /* Verify the comment is a Novell node */
                              /* event comment  */
                              if (3 == NWWordSwap(note->commentType) ||
                                  4 == NWWordSwap(note->commentType) ||
                                  5 == NWWordSwap(note->commentType))
                              {
                                 /* Swap the byte order to Intel format */
                                 node->net    = NWLongSwap(node->net);
                                 node->nodeHi = NWLongSwap(node->nodeHi);
                                 node->nodeLo = NWWordSwap(node->nodeLo);
                              }

                              PrintNoteRecord(format, note);
                              printf("\n\n");
                           }
                           else
                              printf("An error occured in GetFormatString\n")

                           break;
                 }
              }
           }
```

```
      /* Wait for a key press */
      printf("Press ESC to exit.  Press SPACEBAR to continue...\n\n");
      if (getch() == 27)      /* ESC */
         break;
      if (kbhit())   /* Just flush the second value in an extended key
         getch();

   } while (!eof(fileno(acct)));


   /* Close our file  */
   fclose( acct );
}


/* Function that reads Net$Rec.Dat and returns the format string  */
nuint   GetFormatString
(
   pnstr     formatString,
   pnuint8   fieldCount,
   pnstr     dataType,
   nuint16   commentType
)
{
   nuint   i;
   nuint   returnValue; /* Variable to hold the value to return
   nint    ccode;       /* Variable to receive return codes
   FILE    *rec;        /* Pointer to the stream associated with Net$Re
   nstr    buffer[255]; /* Buffer to read the record from Net$Rec.Dat i
   nint    formatStringLength;
                        /* Variable to hold the length of the format st

   FormatRecord   *format;


   /* Initialize returnValue */
   returnValue = -1;        /* Failure */


   /* Open the stream for Net$Rec.Dat */
   if (access("NET$ACCT.DAT", 04))
      rec = fopen("NET$REC.DAT", "rb");   /*  File is in default direct
   else
      rec = fopen("SYS:PUBLIC\NET$REC.DAT", "rb"); /* Not in default di

   if (NULL == rec)
   {
      printf("\tUnable to open Net$Rec.Dat\n");
      return returnValue;
   }
```

```
/* Overlay the format record structure on top of buffer  */
format = (FormatRecord *)buffer;

do
{
   /* Let's read the length of the next record from Net$Rec.Dat  */
   ccode = read(fileno(rec), &format->length, 2);
   format->length = NWWordSwap(format->length);
   printf("Net$Rec.Dat Record length = %d \n", format->length);

   /* Verify read was successful and that length is smaller */
   /* than buffer sizen                                      */
   if (2 == ccode  &&  253 > format->length)
   {
      /* Read the rest of the record from Net$Rec.Dat  */
      ccode = read(fileno(rec), &format->commentType,
              format->length);

      /* Verify read was successful  */
      if (format->length != ccode)
         break;

      /* Check for a match on commentType  */
      if (NWWordSwap(format->commentType) == commentType)
      {
         /* Set the value to return in fieldCount  */
         *fieldCount = format->fieldCount;

         /* Fill the dataType array  */
         for (i = 0; i < *fieldCount; i++)
            dataType[i] = format->firstDataType[i];

         /* The format string is a length preceeded string */
         /* Get the length of the string */
         formatStringLength = format->firstDataType[*fieldCount];

         /* Fill the formatString  */
         strcpy(formatString,
            (nstr *)&format->firstDataType[*fieldCount + 1]);

         /* Terminate the formatString */
         formatString[formatStringLength] = 0;

         /* Return Success  */
         returnValue = 0;            /* Success */
         break;
      }
   }
} while (!eof(fileno(rec)));

/* Close our file  */
fclose( rec  );
```

```
   return returnValue;
}


/* Function that prints a note record to screen  */
int   PrintNoteRecord
(
   pnstr        format,
   NoteRecord   *note
)
{
   int       count;
   va_list   argList;


   va_start(argList, note->comment - 1);
   count = vprintf(format, argList);
   va_end(argList);

   return count;
}


/* Function that prints a Connect Time Charge record to screen  */
int PrintConnectTimeRecord
(
   pnstr           format,
   ChargeComment   *comment,
   double          bytesRead,
   double          bytesWritten
)
{
   return( printf(format, comment->unit1, comment->unit2,
                  bytesRead, bytesWritten) );
}


/* Function that prints a charge record to screen  */
int   PrintChargeRecord
(
   pnstr          format,
   ChargeRecord   *record
)
{
   int       count;
   va_list   argList;

   va_start(argList, record->comment - 1);
   count = vprintf(format, argList);
   va_end(argList);
   return count;
}
```

Parent Topic:

NetWare Server Charges

Accounting: Concepts

Account Locked Comment

An *account locked* comment records the network node on which a user's account has been locked because of too many failed login attempts. The following table shows the format of an account locked comment.

Offset	Field	Type	Order
0	Net	nuint32	high-low
4	Node (hi)	nuint32	high-low
8	Node (lo)	nuint16	high-low

Balances of Accounting

When accounting is enabled on a NetWare® server, users are given a global account balance. Expenditures for all accounting-enabled services are charged against this value. The account balance is stored in the user's ACCOUNT_BALANCE property. The following table shows fields within this property.

Table auto. The ACCOUNT_BALANCE Property

Offset	Field	Type	Order
0	balance	nint32	high-low
4	credit limit	nint32	high-low
8	reserved	nuint8[120]	

The *balance* field contains a signed long integer indicating the current balance.

The *credit limit* field contains a signed long integer indicating the lowest permissible balance. Requests are denied for services causing the value in *balance* to fall below the value in *credit limit*. *credit limit* can be positive (maintaining a minimum balance) or negative (permitting the balance to fall below 0).

Charge Record Auditing

A charge record is created each time a charge is placed against a user balance. The following table shows the format of a charge record:

Table auto. Charge Record Format

Off set	Field	Type	Order
0	length	nuint16	high-low
2	server ID	nuint32	high-low
6	time stamp	nuint8[6]	
12	record type	nuint8[1]	
13	completion code	nuint8	
14	service type	nuint16	high-low
16	client ID	nuint32	high-low
20	amount	nuint32	high-low
24	comment type	nuint16	high-low
26	comment	variabl e	

You can identify the record type by checking the *record type* field. Identify the record type as follows:

charge record: Record type = 1

note record: Record type = 2

The *length* field contains the total length of the remaining fields in the record (that is, record length minus 2 bytes).

The *time stamp* field contains the date and time the charge was submitted. The format is year (year=current year minus 1900), month, day, hour, minute, second, where each is a byte-sized integer.

The *server ID* and *service type* fields identify the object submitting the charge and the type of service. The server ID is a bindery object ID. The service type is the bindery object type of the server. You must provide your service type when you submit the charge. The amount is the amount being charged to the user.

The *client ID* field identifies the user being charged for the service. Although this value identifies the user by bindery object ID, you need to know only the name of the user to submit the charge request. For more information about comment type and the format of the comment field, see Event Comments of Accounting.

The *amount* field is a long unsigned integer indicating the amount of the charge.

The *comment* and *comment type* fields supply information specific to the service. Novell defines standard comment types and formats for recording this information. The length of the *comment* field can be determined by the type of the comment. The *completion code* field is set to 0 if the request is successful.

Chargeable Services

Services are charged for every half hour from the time the user logs in until the user logs out. If during this period the user account balance is found to have fallen below the credit limit, the user is given a five minute and a one minute warning, and is then disconnected. Chargeable services include connection time, number of requests, number of blocks read, and number of blocks written. The rates for these services are stored in the following properties:

CONNECT_TIME

REQUESTS_MADE

BLOCKS_READ

BLOCKS_WRITTEN

Although there is a separate property for each service, the properties share the same format.

The following table shows the format of service rate properties:

Table auto. Format for Service Rate Properties

Off set	**Field**	**Type**	**Order**
0	time of next charge	nuint 32	high-low
4	current charge rate multiplier	nuint 16	high-low
6	current	nuint	high-low

6	current charge rate divisor	nuint 16	high-low
8	days charge occurs mask	nuint 8	
9	time charge occurs	nuint 8	
10	charge rate multiplier	nuint 16	high-low
12	charge rate divisor	nuint 16	high-low

NOTE: Service rate properties can store up to 20 charge rates. Each charge rate includes the following four fields (6 bytes):

days charge occurs mask
time charge occurs
charge rate multiplier
charge rate divisor

The first charge rate begins at offset 8, the second charge rate begins at offset 14 and the third charge rate begins at offset 20. The following table shows the information for the second and third charge rates, if used. Charge rates 4 through 20 would follow this pattern if used.

Table auto. Second and Third charge rates, if used

Off set	**Field**	**Type**	**Order**
14	days charge occurs mask	nuint 8	
15	time charge occurs	nuint 8	
16	charge rate multiplier	nuint 16	high-low
18	charge rate divisor	nuint 16	high-low
20	days charge	nuint 8	

	occurs mask		
21	time charge occurs	nuint 8	
22	charge rate multiplier	nuint 16	high-low
24	charge rate divisor	nuint 16	high-low

The first three fields in the service rate property maintain the **current** charge rate. The time of next charge field contains the time when the current charge is replaced with a different rate. The current charge rate multiplier and current charge rate divisor are used to compute the charges in the following manner:

1. The unit of service (total connect time, number of requests performed, number of blocks read, number of blocks written) is multiplied by the current charge rate multiplier.
2. The product from Step 1 is divided by the current charge rate divisor, and the result is deducted from the user's account balance.

Each group of four fields after the first three fields contain a charge rate. The days charge occurs mask and time charge occurs together indicate when the charge should take effect. The days charge occurs mask indicates the day of the week (bit 0=Sunday,..., bit 6=Saturday) and time charge occurs indicates the time of day on the half hour (0=12am,...,47=11:30pm).

Each rate has its own charge rate multiplier and divisor. The server moves these values into current charge rate multiplier and current charge rate divisor when the charge rate becomes effective.

Connect Time Comment

A connect time comment records NetWare® server usage. It includes the number of minutes the workstation was connected to the server, the number of packets sent to the server, and the number of bytes written and read. The following table shows the format of a connect time comment.

Table auto. Connect Time Comment Format

Off set	Field	Type	Order
0	connect time	nuint 32	high-low

4	request count	nuint 32	high-low
8	bytes read (hi)	nuint 16	high-low
10	bytes read (middle)	nuint 16	high-low
12	bytes read (lo)	nuint 16	high-low
14	bytes written (hi)	nuint 16	high-low
16	bytes written (middle)	nuint 16	high-low
18	bytes written (lo)	nuint 16	high-low

Parent Topic:

Event Comments of Accounting

Disk Storage Charges

Disk storage charges are levied every half hour. The disk storage charge rates are stored in the DISK_STORAGE property attached to theNetWare® server. The format for this property varies slightly from the service properties due to the static nature of storage. Charges are computed for all users during the same period (independent of user login sessions). The following table shows this format:

Table auto. The DISK_STORAGE Property

Off set	**Field**	**Type**	**Order**
0	time of next charge	nuint 32	high-low
4	current charge rate multiplier	nuint 16	high-low
6	current charge rate divisor	nuint 16	high-low
8	days charge occurs mask	nuint 8	

9	time charge occurs	nuint 8	
10	charge rate multiplier	nuint 16	high-low
12	charge rate divisor	nuint 16	high-low

Like service rate properties, the disk storage property can store up to 20 charge rates. The first charge rate begins at offset 8 and includes 4 fields (6 bytes). The purpose of each field is exactly the same as for a service rate property (See Chargeable Services). The current rate for disk storage is computed in the following manner for each user:

1. The current disk storage is calculated in 4K blocks.
2. The total blocks occupied by a user are multiplied by the number of half hours since the last disk storage charge was made.
3. The total half-hour blocks are multiplied by the current charge rate multiplier.
4. The product from Step 3 is divided by the current charge rate divisor, and the result is deducted from the user's account balance.

The time of next charge and time of previous charge are used to calculate when the next charge rate takes effect.

Disk Storage Comment

A disk storage comment records the number of blocks owned by the user and the amount of time the blocks have been in the user's possession. The time is expressed in half hour increments. The following table shows the format of a disk storage comment:

Table auto. Disk Storage Comment Format

Off set	Field	Type
0	number of blocks owned	nuint32
4	number of half hours	nuint32

Elements of Accounting

The following elements compriseNetWare® accounting:

Servers of Accounting

Balances of Accounting

Events of Accounting

Holds of Accounting

The accounting system defines a special bindery property to manage each element.

Event Comments of Accounting

A comment is a field in a charge or note record that stores information specific to the service submitting the charge or note. For example, if you are charging for disk storage, use a disk storage comment. There are six predefined comment types. If one of these doesn't serve your purpose, contact Novell's Developer Relations (see the preface for phone numbers) to request a unique comment type. Standard comment types and their values include the following:

1 = Connect time charge
2 = Disk storage charge
3 = Log in note
4 = Log out note
5 = Account locked note
6 = Server time modified note

Comments are embedded in charge or note records and supplement the standard information that appears there. For example, a client ID of the record identifies the user associated with the comment.

Related Topics:

Connect Time Comment

Disk Storage Charges

Login Comment

Logout Comment

Account Locked Comment

Server Time Modified Comment

Connect Time Comment

Event Comments of Accounting Format File

A comment format file, net$rec.dat, found in the sys:system directory, serves as a companion to the audit file. This file is for information and ease-of-use only and cannot be written to. (Do not delete this file,NetWare® does not recreate it as it does the net$acct.dat file.) The file allows you to display information from the *comment* field in a charge or note record. For example, a disk storage comment indicates the number of disk blocks owned by a user over a period of half hours. If the user owns 25 blocks for 10 half hours, the *comment* field stores only the numeric values 25 and 10. A record in the net$rec.dat file stores a standard format control string for displaying this data. In this example, the format string is

```
"%lu disk blocks stored for %lu half-hours."
```

This string is intended to fit conveniently into a *printf*-style statement, as in the following code:

```
#include<nwmisc.h>
nuint32   blocks, time;

printf("%lu disk blocks stored for %lu half-hours.", NWLongSwap(blocks)
```

In your code, the format control string would not appear as a literal value but as a pointer to the area into which you copy the formatting string from the comment formatting file. Since values in the comment are stored in high-low order, they are swapped in the example above.

Events of Accounting

Accounting includes a rudimentary auditing service that allows you to record accounting events as they occur. (Accounting audits are separate from the full-scale services provided by Auditing.) When auditing is enabled on a server an accounting audit file, net$acct.dat, is created and put in the file server’s sys:system directory. net$acct.dat stores the auditing record. Standard file I/O functions can manipulate this file, which has normal file attributes. The NetWare® PAUDIT utility reads the information from this file to the standard output.

The accounting audit file contains records of two types: charge records and note records. A charge record is added to the file when a user’s account balance is charged for services. Note records are added at the discretion of the services using accounting. A note is purely informational and has no effect on any account balances.

The net$acct.dat file grows in size as accounting events occur. This file should be monitored regularly and archived and deleted when it gets too big. Once the net$acct.dat file has been archived and deleted, NetWare will automatically recreate it the next time the user logs in.

Holds of Accounting

Account holds on user balances reserve funds for services the server supplies. Holds allow verification that current funds can cover particular requests. Account holds are not required, but your application should at least check *balance* in the ACCOUNT_BALANCE property to verify funding for the specific request.

Holds are recorded in the user property, ACCOUNT_HOLDS. This property is present only if there is a hold on the account. The following table shows the format of the ACCOUNT_HOLDS property.

Table auto. The ACCOUNT_HOLDS Property

Offset	Field	Type	Order
0	object ID	nint32	high-low
4	amount	nuint32	high-low

Each hold consists of an *object ID* and an *amount* field. The *object ID* field is the ID of the object placing the hold. The *amount* field is the amount of the hold. Each of the two parameters uses 4 bytes. The maximum size of the property is 128 bytes; therefore, ACCOUNT_HOLDS can have no more than 16 holds placed on the account at once. If the object ID is 0, the*amount* field is not meaningful. HOLDS_STATUS and HOLDS_INFO are defined to handle this data.

Introduction to Accounting

Accounting is a bindery-based service that must be enabled on a server running NetWare® 2.1 and above before the services can be used. Accounting contains functions required for administering and recording accounting events. Call Accounting functions, for example, in time sharing situations and for tracking print server and database usage, connection time, and disk storage.

Developers can also take advantage of accounting to charge for services and to audit events. Because Accounting relies heavily on the bindery, a familiarity with bindery operations is necessary before reading about Accounting.

Login Comment

A login comment records the network node of a workstation that the user logged in from. The NetWare® server uses this comment as an auditing note and not as a charge. The following table shows the format of a login comment:

Off set	Field	Type	Order
0	Net	nuint 32	high-low
4	Node (hi)	nuint 32	high-low
8	Node (lo)	nuint 16	high-low

Logout Comment

A logout comment records the network node of a workstation the user has logged out from. The NetWare® server uses this comment as an auditing note and not as a charge. The following table shows the format of a logout comment:

Off set	Field	Type	Order
0	Net	nuint 32	high-low
4	Node (hi)	nuint 32	high-low
8	Node (lo)	nuint 16	high-low

NetWare Server Charges

NetWare® servers use the accounting system to charge for services and disk storage. The supervisor determines the charge rates, typically by using the SYSCON utility (NetWare 3.x) or NETADMIN and NWADMIN utilities (NetWare 4.x). Rates are stored as bindery properties attached to the server object. NetWare servers handle disk storage accounting somewhat differently from other services, so service and storage are considered separately in this section.

Note Record Auditing

The note record shares many of the same fields found in the change record. The main difference between them is that the note record does not include a charge amount. Notes are used to record events for the system administrator, such as the time a user logged in to the server. The following table shows the format of a note record:

Table auto. Note Record Format

Off set	**Field**	**Type**	**Order**
0	length	nuint16	high-low
2	server ID	nuint32	high-low
6	time stamp	nuint8 [6]	
12	record type	nuint8 [1]	
13	reserved	nuint8	
14	service type	nuint16	high-low
16	client ID	nuint32	high-low
20	comment type	nuint16	high-low
22	comment	variable	

Server Time Modified Comment

A server time modified comment records a modification in the system time of the NetWare server. The comment contains the time value before the change was made. The following table shows the format of a server time modified comment:

Table auto. Server Time Modified Comment Format

Off set	**Field**	**Type**
0	year since 1900	nuint8

1	month	nuint8
2	day	nuint8
3	hour	nuint8
4	minute	nuint8
5	second	nuint8

Servers of Accounting

Any object permitted to charge for services through the accounting service qualifies as an accounting server. A server receives accounting status by recording its bindery object ID in the ACCOUNT_SERVERS property of the NetWare server. Your application needs supervisor equivalence to add itself to this property. (For more information about adding an application to the ACCOUNTING_SERVERS property, see **NWAddObjectToSet**.)

Standard Format Control Strings

The following is a list of the format control strings for standard Novell comment types:

Connect Time Format:

```
"Connected %lu minutes: %lu requests; %04x%04x%04xh
bytes read; %04x%04x%04xh bytes written"
```

Disk Storage Format:

```
"%lu disk blocks stored for %lu half-hours."
```

Login Format:

```
"Login from address %lx:%lx%x."
```

Logout Format:

```
"Logout from address %lx:%lx%x."
```

Account Locked Format:

```
"Account intruder lockout caused by address:
%lx:%lx%x."
```

Server Time Modified Format:

```
"System time changed to 19%02d-%02d-%02d%d:%02:%02d."
```

String Formatting Records

The following table shows the format of a record in the net$rec.dat file.:

Table auto. String Formatting Record

Off set	Field	Type	Order
0	length	nuint16	high-low
2	comment type	nuint16	high-low
4	field count	nuint8[1]	
5	data type	nuint8[field count]	
?	buffer	variable	

Use the “comment type” field to associate a formatting record in this file with a comment in the audit file. For example, the formatting record for disk storage comments would have a value of 2 in comment type.

length is the total length of the remaining fields in the record (i.e., record length - 2 bytes).

data type identifies the data types found sequentially in the associated comment. The following values are defined:

1 = nuint8
2 = nuint16
3 = nuint32
4 = TEXT

Text data is a length-preceded string of printable characters.

The number of data types is stored in the field count field.

Below are two examples of how the *buffer* field is formatted. The first table is for a Connect Time Charge with a Field Count of four. The second table is for a Disk Storage Charge with a Field Count of two.

Table auto. Connect Time Charge

Off set	Field	Val ue	Type
0	data type 1	3	nuint8
1	data type 2	3	nuint8
2	data type 3	5	nuint8

3	data type 4	5	nuint8
4	data value 1		nuint32
8	data value 2		nuint32
12	data value 3		nfloat48
18	data value 4		nfloat48
24	length of string	81	nuint8
25	printable string		ASCII

Table auto. Disk Storage Charge

Offset	**Field**	**Value**	**Type**
	data type 1	3	nuint8
1	data type 2	3	nuint8
2	data value 1		nuint32
6	data value 2		nuint32
10	length of string	55	nuint8
11	printable string		ASCII

Accounting: Functions

NWGetAccountStatus

Returns the account status of a NetWare® server Bindery or Directory Services object including its balance, credit limit, and holds

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: Accounting

Syntax

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

NWCCODE N_API NWGetAccountStatus (
   NWCONN_HANDLE         conn,
   nuint16               objType,
   pnstr8                objName,
   pnint32               balance,
   pnint32               limit,
   HOLDS_STATUS N_FAR   *holds);
```

Pascal Syntax

```
#include <nwacct.inc>

Function NWGetAccountStatus
  (conn : NWCONN_HANDLE;
   objType : nuint16;
   objName : pnstr8;
   balance : pnint32;
   limit : pnint32;
   Var holds : HOLDS_STATUS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objType

(IN) Specifies the type of the object for which the status is desired.

objName

(IN) Points to a string containing the name of the object for which the accounting status is desired. (48 characters maximum or will be truncated.)

balance

(OUT) Points to the amount of value units available to the object to buy services on the network (optional, pass in NULL).

limit

(OUT) Points to the lowest level the object's account balance can reach before the object can no longer buy services on the network (optional, pass in NULL).

holds

(OUT) Points to HOLDS_STATUS containing a list of objects placing a hold on the object's account (optional, pass in NULL).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89C0	NO_ACCOUNTING_PRIVILEGES
0x89C1	LOGIN_DENIED_NO_ACCOUNT_BALANCE
0x89C4	ACCOUNTING_DISABLED
0x89E8	NOT_ITEM_PROPERTY
0x89EA	NO_SUCH_MEMBER
0x89EB	NOT_GROUP_PROPERTY
0x89EC	NO_SUCH_SEGMENT
0x89EF	INVALID_NAME
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

NWGetAccountStatus queries a NetWare server's Bindery or Directory bindery context for the current account status of a specified object by passing the object name and type. **NWGetAccountStatus** returns the object's balance, limit and holds.

balance contains the object's account balance, usually in some established monetary unit such as cents.

holds lists servers calling **NWSubmitAccountHold** against the object and the amount reserved by each value-added server. *holds* also lists the object ID number of a value-added server calling **NWSubmitAccountHold** against the object. Up to 16 servers can place holds on the account at one time. Multiple holds from the same server are combined. Each server hold is made up of two fields: (1) the object ID of the server placing the hold, and (2) the amount of the server's hold.

The ACCOUNT_SERVERS property on the file server must contain the object ID of the requesting server. Otherwise, **NWSubmitAccountHold** will return NO_ACCOUNTING_PRIVILEGES. The user must have an ACCOUNT_BALANCE property on the NetWare server or **NWSubmitAccountHold** will return LOGIN_DENIED_NO_ACCOUNT_BALANCE.

NCP Calls

0x2222 23 150 Get Current Account Status

See Also

NWQueryAccountingInstalled, **NWSubmitAccountCharge**, **NWSubmitAccountHold**, **NWSubmitAccountNote**

NWQueryAccountingInstalled

Determines whether accounting is installed and/or enabled on 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: Accounting

Syntax

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

NWCCODE N_API NWQueryAccountingInstalled (
   NWCONN_HANDLE   conn,
   pnuint8         installed);
```

Pascal Syntax

```
#include <nwacct.inc>

Function NWQueryAccountingInstalled
   (conn : NWCONN_HANDLE;
   installed : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

installed

(OUT) Points to the installed value. It returns 1 if accounting is installed on the NetWare server; otherwise, *installed* returns 0.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

0x8811	INVALID_SHELL_CAL
0x8836	INVALID_PARAMETER

Remarks

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

When accounting is enabled, the NetWare server object has the property ACCOUNT_SERVERS.

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 23 60	Scan Property
0x2222 104 1	Ping for NDS NCP

NWSubmitAccountCharge

Charges an amount against an object's balance and either relinquishes a hold or reduces the hold by the charge amount

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: Accounting

Syntax

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

NWCCODE N_API NWSubmitAccountCharge (
   NWCONN_HANDLE   conn,
   nuint16         objType,
   pnstr8          objName,
   nuint16         serviceType,
   nint32          chargeAmt,
   nint32          holdCancelAmt,
   nuint16         noteType,
   pnstr8          note);
```

Pascal Syntax

```
#include <nwacct.inc>

Function NWSubmitAccountCharge
  (conn : NWCONN_HANDLE;
   objType : nuint16;
   objName : pnstr8;
   serviceType : nuint16;
   chargeAmt : nint32;
   holdCancelAmt : nint32;
   noteType : nuint16;
   note : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objType

(IN) Specifies the type of the object to be charged.

objName

(IN) Points to a string containing the name of the object whose account is to be charged (48 character maximum or will be truncated).

serviceType

(IN) Specifies the type of service for which the charge is being made.

chargeAmt

(IN) Specifies the amount to be charged to the object account balance.

holdCancelAmt

(IN) Specifies the amount the hold will be reduced (set to zero if there are no holds).

noteType

(IN) Specifies the note type to be written to the audit report.

note

(IN) Points to a note associated with the charge and stored as an audit record (255 character maximum or will be truncated).

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
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8996	SERVER_OUT_OF_MEMORY
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89C0	NO_ACCOUNTING_PRIVILEGES
0x89C1	LOGIN_DENIED_NO_ACCOUNT_BALANCE
0x89C2	LOGIN_DENIED_NO_CREDIT
0x89C4	ACCOUNTING_DISABLED
0x89E8	WRITE_PROPERTY_TO_GROUP
0x89EA	NO_SUCH_MEMBER
0x89EB	NOT_GROUP_PROPERTY
0x89EC	NO_SUCH_SEGMENT
0x89EF	INVALID_NAME
0x89F0	WILD_CARD_NOT_ALLOWED

0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

NWSubmitAccountCharge can write a note about the transaction in an audit record (optional). The charge and hold amounts do not have to be the same.

objType and *objName* must uniquely specify the object and cannot contain wildcard characters.

serviceType usually contains the object type of the charging account server. The common server object types are listed below:

Object	**Type**	**High-Low Format**
Job Server	OT_JOB_SERVER	0x0500
Print Server	OT_PRINT_SERVER	0x0700
Archive Server	OT_ARCHIVE_SRVER	0x0900

noteType contains the number of the note type. Note types are administered by Novell®, Inc. and are listed below:

Note	**Description Type**
1	Connect time charge
2	Disk storage charge
3	Log in note
4	Log out note
5	Account locked note
6	Server time modified note

Developers should contact Novell for unique note types. Note types greater than 8000H are reserved.

NOTE: *note* is the entry the value-added server makes in SYS:SYSTEM\NET$ACCT.DAT.

NCP Calls

0x2222 23 151   Submit Account Charge

See Also

NWSubmitAccountHold, **NWSubmitAccountNote**

NWSubmitAccountHold

Reserves a specified amount of an object's account balance before the object receives and is charged for a service on the network

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: Accounting

Syntax

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

NWCCODE N_API NWSubmitAccountHold (
   NWCONN_HANDLE   conn,
   nuint16         objType,
   pnstr8          objName,
   nint32          holdAmt);
```

Pascal Syntax

```
#include <nwacct.inc>

Function NWSubmitAccountHold
  (conn : NWCONN_HANDLE;
   objType : nuint16;
   objName : pnstr8;
   holdAmt : nint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objType

(IN) Specifies the type of the object for which the hold is desired.

objName

(IN) Points to the name of the object for which the hold is desired (48 character maximum).

holdAmt

(IN) Specifies the amount to be held against the object's account balance.

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
0x8994	NO_WRITE_PRIVILEGES_OR_READONLY
0x8996	SERVER_OUT_OF_MEMORY
0x89A2	READ_FILE_WITH_RECORD_LOCKED
0x89C0	NO_ACCOUNTING_PRIVILEGES
0x89C1	LOGIN_DENIED_NO_ACCOUNT_BALANCE
0x89C2	LOGIN_DENIED_NO_CREDIT
0x89C3	ERR_TOO_MANY_HOLDS
0x89C4	ACCOUNTING_DISABLED
0x89E8	WRITE_PROPERTY_TO_GROUP
0x89EA	NO_SUCH_MEMBER
0x89EB	NOT_GROUP_PROPERTY
0x89EC	NO_SUCH_SEGMENT
0x89EF	INVALID_NAME
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FC	NO_SUCH_OBJECT
0x89FE	BINDERY_LOCKED
0x89FF	HARDWARE_FAILURE

Remarks

NWSubmitAccountHold reserves a specified amount of an object's account balance that object receives and is charged for services on the network.

objType and *objName* must uniquely identify the object and cannot contain wildcard characters.

holdAmt gets the amount the server expects to charge for the service it is about to provide to the object.

NOTE: No more than 16 servers can reserve amounts of an object's

NOTE: No more than 16 servers can reserve amounts of an object's account balance at one time. Multiple holds from the same server are combined.

NCP Calls

0x2222 23 152   Submit Account Hold

See Also

NWQueryAccountingInstalled, **NWGetAccountStatus**, **NWSubmitAccountCharge**, **NWSubmitAccountNote**

NWSubmitAccountNote

Adds a note about an accounting transaction to an audit record

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: Accounting

Syntax

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

NWCCODE N_API NWSubmitAccountNote (
   NWCONN_HANDLE   conn,
   nuint16         objType,
   pnstr8          objName,
   nuint16         serviceType,
   nuint16         noteType,
   pnstr8          note);
```

Pascal Syntax

```
#include <nwacct.inc>

Function NWSubmitAccountNote
  (conn : NWCONN_HANDLE;
   objType : nuint16;
   objName : pnstr8;
   serviceType : nuint16;
   noteType : nuint16;
   note : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objType

(IN) Specifies the type of the object for which the note is being submitted.

objName

(IN) Points to the name of the object to receive the note (48 character maximum or will be truncated).

serviceType

(IN) Specifies the object type of the charging account server.

noteType

(IN) Specifies the note type (255 character maximum or will be truncated).

note

(IN) Points to a note to be added to an audit record.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8901	ERR_INSUFFICIENT_SPACE
0x8996	SERVER_OUT_OF_MEMORY
0x89C0	NO_ACCOUNTING_PRIVILEGES
0x89C1	LOGIN_DENIED_NO_ACCOUNT_BALANCE
0x89C4	ACCOUNTING_DISABLED
0x89E8	WRITE_PROPERTY_TO_GROUP
0x89EA	NO_SUCH_MEMBER
0x89EB	NOT_GROUP_PROPERTY
0x89EC	NO_SUCH_SEGMENT
0x89EF	INVALID_NAME
0x89F0	WILD_CARD_NOT_ALLOWED
0x89FC	NO_SUCH_OBJECT
0x89FF	HARDWARE_FAILURE

Remarks

objType and *objName* must uniquely identify the object and cannot contain wildcard characters.

serviceType usually contains the object type of the charging account server. The common server object types are listed below:

Object	**Type**	**High-Low Format**
`Job Server`	`OT_JOB_SERVER`	`0x0500`
`Print Server`	`OT_PRINT_SERVER`	`0x0700`

```
Archive Server   OT_ARCHIVE_SRVER    0x0900
```

noteType contains the number of the note type. Note types are administered by Novell and are listed below:

Note	**Description Type**
1	Connect time charge
2	Disk storage charge
3	Log in note
4	Log out note
5	Account locked note
6	Server time modified note

Developers should contact Novell for unique note types. Note types greater than 8000H are reserved.

NOTE: *note* contains the entry the server adds to the audit record. The audit record is contained in SYS:SYSTEM\NET$ACCT.DAT.

NCP Calls

0x2222 23 153 Submit Account Note

See Also

NWSubmitAccountCharge, **NWSubmitAccountHold**

Accounting: Structures

HOLDS_INFO

Defines information for an object that places a hold on an account

Service: Accounting

Defined In: nwacct.h

Structure

```
typedef struct
{
   nuint32   objectID;
   nint32    amount;
} HOLDS_INFO;
```

Pascal Structure

```
Defined in nwacct.inc

   HOLDS_INFO = Record
      objectID : nuint32;
      amount : nint32
   End;
```

Fields

objectID

Indicates the object ID of the server placing the hold. Multiple holds from the same server are combined.

amount

Indicates the amount of the associated *objectID*'s hold.

HOLDS_STATUS

Stores a list of holds placed on an account

Service: Accounting

Defined In: nwacct.h

Structure

```
typedef struct
{
   nuint16      holdsCount;
   HOLDS_INFO   holds[16];
} HOLDS_STATUS;
```

Pascal Structure

```
   HOLDS_STATUS = Record
      holdsCount : nuint16;
      holds : Array[0..15] Of HOLDS_INFO
   End;
```

Defined In

nwacct.inc

Fields

holdsCount

Indicates the number of *NW_HOLDS* in *holds* array.

holds

Indicates a list of servers calling **NWSubmitAccountHold** against an object and the amount reserved by each value-added server.

Application Launcher

Application Launcher: Guides

Application Launcher: General Guide

Tasks

Using the Application Launcher

Using the Application Launcher Example

Concepts

Application Launcher Introduction

Related Topics:

Application Launcher: Tasks

Application Launcher: Examples

Application Launcher: Concepts

Application Launcher: Functions

Application Launcher: Structures

Parent Topic:

Management Overview

Application Launcher: Tasks

Using the Application Launcher

To enable users to launch a Directory Services application do the following:

1. **Call NWAPPGetFolders to retrieve a list of all the application objects available to the user through Directory Services.**

2. **For each application displayed in the list call NWAPPGetObjectAttributes and NWAPPFreeGetObjectAttributes.**

   This retrieves the icon and a description of each application for display on the screen.

3. **Call NWAPPFreeGetFolders to clean up the memory allocated by NWAPPGetFolders.**

4. **Start a timer that calls NWAPPMonitorApplications (about once a second) to check if any launched applications have terminated.**

   The resources for a closed application will automatically be cleaned up.

5. **When a user launches an application, call NWAPPLaunchApplication .**

   All necessary resources for the application will be allocated (drive mappings and port captures for example).

Parent Topic:

Application Launcher: General Guide

Related Topics:

Using the Application Launcher Example

Application Launcher: Examples

Using the Application Launcher Example

The following example gets the list of applications for the current user and displays them on the screen.

```
void NWAPPExample()
{
      APP_DIRECTORY_OBJECT *c1;
      APP_CONTEXT_HANDLE   context;
      char                 tStr[255];

      printf("Starting...\n\n");

      /* Allocate a context handle. */
      context = NWAPPAllocContextHandle();

      /* Check if this tree has any application objects. */
      if (NWAPPIsSchemaModified(context) == FALSE) {
            printf("Schema is not modified!\n");
            return;
      }

      /* Show all of the desktop groups for this user. */
      printf("-----------------\n");
      printf("Desktop Groups...\n");
      printf("-----------------\n");
      APP_DESKTOP_FOLDER *f2, *c2;
      f2 = NWAPPGetFolders(context, APP_DESKTOP, 5, NULL);
      c2 = f2;
      while (c2 != NULL) {
            printf("Full object name: %s\n", c2->name);
            NWAPPStripFullObjectName(context, c2->name, tStr);
            printf("Stripped object name: %s\n", c2->name);
            c1 = c2->head;
            while (c1 != NULL) {
                  printf("\t%s (%s)\n", c1->name, c1->fullName);
                  c1 = c1->next;
            }
            c2 = c2->next;
      }
      printf("\n\n");

      // Launch the first application in the list.
```

```
        nint16 rslt;
        nint16 launchError;
        char launchErrStr[255];
        nint16 id;
        if (f2 != NULL) {
              rslt = NWAPPLaunchApplicationId(context,f2->head->fullName,
                                               &launchError,launchErrStr);
              if (rslt != APP_SUCCESS) {
                    printf("\n\n");
                    printf("ERROR: NWAPPLaunchApplication failed - 0x%04X
                    printf("       launchError  = 0x%04X\n", launchErrStr
                    printf("       launchErrStr = %s\n", launchErrStr);
              }
        }

        /* NWAPPMonitorApplicationId should be called here until the */
        /* program hasbeen terminated. */

        /* Free up the memory allocated by NWAPPGetFolders. */
        NWAPPFreeGetFolders(f2);

        /* Cleanup. */
        NWAPPFreeContextHandle(context);
  }
```

Parent Topic:

Using the Application Launcher

Application Launcher: Concepts

Application Launcher Introduction

NetWare® Application Launcher enables you to integrate Novell® Directory Services™ Application Objects into your application. The Application Launcher also simplifies the retrieval of object attributes for any Directory Services object.

Parent Topic:

Application Launcher: General Guide

Application Launcher: Functions

NWAPPAllocContextHandle

Creates a context handle

Platform: Windows* 3.1, Windows*95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

APP_CONTEXT_HANDLE N_API N_EXPORT NWAPPAllocContextHandle (
   void);
```

Return Values

If successful, **NWAPPAllocContextHandle** returns a context handle. Otherwise, it returns APP_ERROR.

Remarks

NWAPPAllocContextHandle creates a context and returns a context handle. You must free the handle by calling **NWAPPFreeContextHandle**
.

See Also

NWAPPFreeContextHandle

NWAPPCreateAppObject

Creates an application object in the Directory tree and sets various attributes and values for the new application object

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

nuint16 N_API N_EXPORT NWAPPCreateAppObject (
   APP_CONTEXT_HANDLE         context,
   pnstr8                     containerName,
   APP_APPLICATION_OBJECT   *appData);
```

Parameters

context

(IN) Specifies the context where the application object will be created.

containerName

(IN) Specifies the DS name of the container in which to place the application object. For example: `".ou=Accounting.o=novell"`

appData

(IN) Points to APP_APPLICATION_OBJECT defining the various attributes and values for the new application object as well as the object name.

Return Values

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

APP_SUCCESS	
E_INVALID_CONTAINER	Container name does not exist relative to the context
E_FILE_NOT_EXE	*path* in appData does not specify an executable
E_BAD_PATH	*path* in appData does not exist
E_OBJECT_ALREADY_EXISTS	An object of the same name as *name* in appData already exists
E_INVALID_DS_OBJECT	One or more of the DS objects passed into the contacts or associations list is not valid

	or does not exist
APP_ERROR	Object could not be created

NWAPPCreateFullObjectName

Creates a fully typed NDS object name
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

pnstr8 N_API N_EXPORT NWAPPCreateFullObjectName (
   APP_CONTEXT_HANDLE   context,
   pnstr8               name,
   pnstr8               fullName);
```

Parameters

context

(IN) Specifies the valid context of the object. NULL is not accepted.

name

(IN) Points to the object name to type.

fullName

(OUT) Points to the typed name of the object. You must allocate enough memory for the name.

Return Values

If successful, **NWAPPCreateFullObjectName** returns a pointer to the full name of the object.

Remarks

NWAPPCreateFullObjectName expands an object name to its fully typed name. For example, `DJanis` becomes `CN=DJanis.O=Novell`.

NWAPPFreeContextHandle

Frees a context handle allocated by **NWAPPAllocContextHandle**
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

void N_API N_EXPORT NWAPPFreeContextHandle (
   APP_CONTEXT_HANDLE   context);
```

Parameters

context

(IN) Specifies the context handle returned by **NWAPPAllocContextHandle** when the context was created.

Return Values

None

See Also

NWAPPAllocContextHandle

NWAPPFreeGetFolders

Frees the memory allocated by **NWAPPGetFolders**

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

APP_DESKTOP_FOLDER NWAPPFreeGetFolders (
   APP_DESKTOP_FOLDER  *head);
```

Parameters

head

(IN) Points to the first node in a list of folders returned by **NWAPPGetFolders**.

Return Values

None

See Also

NWAPPGetFolders

NWAPPFreeGetObjectAttributes

Frees the memory allocated by **NWAPPGetObjectAttributes**
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

void N_API N_EXPORT NWAPPFreeGetObjectAttributes (
   APP_OBJECT_ATTRIBUTE  *head);
```

Parameters

head

(IN) Points to the first node in the attribute list returned by **NWAPPGetObjectAttributes**.

Return Values

None

See Also

NWAPPGetObjectAttributes

NWAPPGetFolders

Creates a linked list of all the folders for the current user, and the objects that reside in them

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

APP_DESKTOP_FOLDER* N_API N_EXPORT NWAPPGetFolders (
   APP_CONTEXT_HANDLE   dContext,
   nint16               folderType,
   nint16               numContainers,
   pnstr8               userName);
```

Parameters

dContext

(IN) Specifies the valid directory context to scan for objects.

folderType

(IN) Specifies the type of list to return: APP_AUTO_START or APP_DESKTOP.

numContainers

(IN) Specifies the number of containers to include in the search. -1 specifies to start searching the Directory tree from the*userName* parameter to the top for folders.

userName

(IN) Points to the user for which to return a list. NULL specifies to return a list for the current user; otherwise,**NWAPPGetFolders** will return a list for the user name passed in. *userName* must be relative to *dContext.*

Return Values

If successful, **NWAPPGetFolders** returns a pointer to the first node in the list of auto start folders. Otherwise, it returns NULL.

Remarks

NWAPPGetFolders returns a list of APP_DESKTOP_FOLDER objects. Each of these desktop folders has a linked list of APP_DIRECTORY_OBJECT nodes.

To get the actual attribute values for a given node, call **NWAPPGetObjectAttributes** using the *fullName* field of APP_DIRECTORY_OBJECT.

See Also

NWAPPFreeGetFolders

NWAPPGetIconFromFile

Reads an icon resource file and creates an icon based on the information
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

HICON N_API N_EXPORT NWAPPGetIconFromFile (
   HINSTANCE   hInst,
   pnstr8      szFileName);
```

Parameters

hInst

(IN) Specifies the handle to the instance of the application.

szFileName

(IN) Specifies the icon resource file.

Return Values

A handle to an icon. The handle will be NULL if an icon cannot be created for any reason.

Remarks

Only the first icon in a given file is read. To modify **NWAPPGetIconFromFile** to read in all the icons in the .ICO file, you must read in the entire icon resource directory, and find the bits to each icon.

NWAPPGetIconFromHandle

Reads an icon from a binary file

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

HICON NWAPPGetIconFromHandle (
   HINSTANCE       hInst,
   NWFILE_HANDLE   fileHandle);
```

Parameters

hInst

(IN) Specifies a handle to a module or application instance.

fileHandle

(IN) Specifies the handle of the binary file to read.

Return Values

If successful, **NWAPPGetIconFromHandle** returns a handle to an icon. Otherwise it returns NULL.

Remarks

Only the first icon in the given file is read. To modify **NWAPPGetIconFromHandle** to read in all the icons in a .ICO file, you must read in the entire icon resource directory and identify the bits to each icon.

See Also

NWAPPReadIcon

NWAPPGetObjectAttributes

Returns a list of attribute names and types for the specified object

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

APP_OBJECT_ATTRIBUTE* N_API N_EXPORT NWAPPGetObjectAttributes (
   APP_CONTEXT_HANDLE    dContext,
   pnstr8                dsName,
   pnstr8               *attrList,
   nint16                attrNum);
```

Parameters

dContext

(IN) Specifies the NDS context of the object. This must be a valid context.

dsName

(IN) Points to the typed name of the object.

attrList

(IN) Points to an array of pointers to strings that indicate which attributes to return. Set this to NULL to get all attributes (set*attrNum* to zero).

attrNum

(IN) Specifies the number of pointers in the *attrList* array. Set to 0 to return all attributes.

Return Values

If successful, **NWAPPGetObjectAttributes** returns a pointer to the first node in a list of APP_OBJECT_ATTRIBUTE structures. Otherwise, it returns NULL.

Remarks

NWAPPGetObjectAttributes returns a list of APP_OBJECT_ATTRIBUTE structures containing a list of APP_ATTRIBUTE_VALUE structures.

dsName is relative to *dContext*.

Call **NWAPPFreeGetObjectAttributes** to free the memory allocated by **NWAPPGetObjectAttributes**.

See Also

NWAPPFreeGetObjectAttributes

NWAPPIconToBitmap

Takes an HICON and returns a HBITMAP
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

HBITMAP N_API N_EXPORT NWAPPIconToBitmap (
   HICON   hIcon);
```

Parameters

hIcon
(IN) Specifies a handle to an icon.

Return Values

A handle to a bitmap.

NWAPPIsSchemaModified

Checks the Directory schema to see if it has been modified for application objects

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

BOOL N_API N_EXPORT NWAPPIsSchemaModified (
   APP_CONTEXT_HANDLE   dContext);
```

Parameters

dContext

(IN) Specifies the context for the tree you want to check.

Return Values

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

TRUE	The schema has application objects defined.
FALSE	The schema does not have application objects defined.

Remarks

If the current tree (relative to *dContext*) has the correct schema, **NWAPPIsSchemaModified** will return TRUE. Otherwise, FALSE is returned.

NWAPPLaunchApplication

Performs all of the necessary actions to launch the specified application

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

nint16 N_API N_EXPORT NWAPPLaunchApplication (
   APP_CONTEXT_HANDLE   dContext;
   pnstr8               dsObject;
   nint16              *errorRslt;
   pnstr8              errorStr);
```

Parameters

dContext

(IN) Specifies the NDS context of the object. This must be a valid context.

dsObject

(IN) Specifies the full Directory Service name of the application to be launched.

errorRslt

(OUT) Returns the NetWare error if a problem occurred.

errorStr

(OUT) Points to a string indicating the error if an error was returned. You must allocate memory for this string (maximum length is 255 characters).

Return Values

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

APP_SUCCESS	Application was launched successfully.
APP_ERROR	Application could not be launched. Problem is unknown.
E_GETTING_ATTRS	Could not retrieve the object attributes.
E_INVALID_DRIVE_PA TH	The path to the executable was invalid.
E_INVALID_QUEUE_PA	The path to the queue was invalid.

E_INVALID_QUEUE_PATH	The path to the queue was invalid.
E_AUTH_FAILED	Authentication failed.
E_INVALID_MAP_DRIVE	Mapping the drive failed.
E_CAPTURE_PORT	The capture failed. The syntax for the capture may be incorrect.
E_CAPTURE_FLAG	The call to set the capture flags failed.
E_CAPTURE_SETTINGS	The capture failed.
E_SCRIPT_FAILED	The pre or post launch script failed to execute.
E_EXEC_FAILED	The application could not be launched.

Remarks

NWAPPLaunchApplication performs all maintenance necessary to start an application including mapping drives, capturing printers, pre- and post-setup or cleanup.

Call **NWAPPMonitorApplications** to track and clean up resources.

When APP_SUCCESS is not returned, *errorStr* and *errorRslt* are set to the following values:

Return Code	***errorRslt* Value**	***errorStr* Value**
E_GETTING_ATTRS	Undefined	Directory Services object name
E_INVALID_DRIVE_PATH	NetWare Return Code	Executable file path
E_INVALID_QUEUE_PATH	NetWare Return Code	Resource for the capture
E_AUTH_FAILED	NetWare Return Code	Name of the server
E_INVALID_MAP_DRIVE	NetWare Return Code	Resource for the mapping
E_CAPTURE_PORT	NetWare Return Code	Capture command
E_CAPTURE_FLAG	NetWare Return Code	Capture command
E_CAPTURE_SETTINGS	NetWare Return Code	Capture command
E_SCRIPT_FAILED	NetWare Return Code	Relative path after the root

E_EXEC_FAILED	Undefined	Executable file path

NWAPPLaunchApplicationId

Performs all of the necessary actions to launch an application and returns a unique ID for the specified application

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

nint16 N_API N_EXPORT NWAPPLaunchApplicationId (
   APP_CONTEXT_HANDLE   dContext;
   pnstr8               dsObject;
   nint16              *uniqueId;
   nint16              *errorRslt;
   pnstr8              errorStr);
```

Parameters

dContext

(IN) Specifies the NDS context of the object. This must be a valid context.

dsObject

(IN) Specifies the full Directory Service name of the application to be launched.

uniqueId

(OUT) Returns a unique ID identifying the application that was launched. This value is used to call **NWAPPMonitorApplicationById**.

errorRslt

(OUT) Returns the NetWare error if a problem occurred.

errorStr

(OUT) Points to a string indicating the error if an error was returned. You must allocate memory for this string (maximum length is 255 characters).

Return Values

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

APP_SUCCESS	Application was launched successfully.
APP_ERROR	Application could not be launched.

	Problem is unknown.
E_GETTING_ATTRS	Could not retrieve the object attributes.
E_INVALID_DRIVE_PATH	The path to the executable was invalid.
E_INVALID_QUEUE_PATH	The path to the queue was invalid.
E_AUTH_FAILED	Authentication failed.
E_INVALID_MAP_DRIVE	Mapping the drive failed.
E_CAPTURE_PORT	The capture failed. The syntax for the capture may be incorrect.
E_CAPTURE_FLAG	The call to set the capture flags failed.
E_CAPTURE_SETTINGS	The capture failed.
E_SCRIPT_FAILED	The pre or post launch script failed to execute.
E_EXEC_FAILED	The application could not be launched.

Remarks

NWAPPLaunchApplicationId performs all maintenance necessary to start an application including mapping drives, capturing printers, pre- and post-setup or cleanup.

Call **NWAPPMonitorApplications** to track and clean up resources.

If **NWAPPLaunchApplicationId** fails, *uniqueId* is undefined.

When APP_SUCCESS is not returned, *errorStr* and *errorRslt* are set to the following values:

Return Code	***errorRslt* Value**	***errorStr* Value**
E_GETTING_ATTRS	Undefined	Directory Services object name
E_INVALID_DRIVE_PATH	NetWare Return Code	Executable file path
E_INVALID_QUEUE_PATH	NetWare Return Code	Resource for the capture
E_AUTH_FAILED	NetWare Return Code	Name of the server
E_INVALID_MAP_DRIVE	NetWare Return Code	Resource for the mapping
E_CAPTURE_PORT	NetWare Return Code	Capture command

E_CAPTURE_FLAG	NetWare Return Code	Capture command
E_CAPTURE_SETTIN GS	NetWare Return Code	Capture command
E_SCRIPT_FAILED	NetWare Return Code	Relative path after the root
E_EXEC_FAILED	Undefined	Executable file path

NWAPPMakeBitmap

Creates a bitmap based on the DIB
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

HBITMAP N_API N_EXPORT NWAPPMakeBitmap (
   HANDLE   hDIB);
```

Parameters

hDIB

(IN) Specifies a handle to the bitmap's DIB information.

Return Values

A handle to a bitmap. NULL is returned if a bitmap cannot be successfully created.

NWAPPMakeCursor

Creates a cursor based on the DIB information returned by **ReadCursor**
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

HCURSOR NWAPPMakeCursor (
   HINSTANCE   ghInst,
   HANDLE      hDIB,
   LPPOINT     lpptHotSpot);
```

Parameters

ghInst

(IN) Specifies a handle to a module or application instance.

hDIB

(IN) Specifies a handle to the cursor's DIB information.

lpptHotSpot

(IN) Points to a point structure indicating the location of the cursor's hot spot.

Return Values

If successful, **NWAPPMakeCursor** returns a handle to a cursor. Otherwise, it returns NULL.

Remarks

The steps involved in creating a cursor from a DIB are similar to those involved in creating an icon from a DIB and include:

1. Obtain a pointer to the cursor's DIB bits.
2. Divide the DIB'd height by 2 to account for the fact that the DIB stores both the XOR and the AND masks, one after the other.
3. Determine the offset of the XOR bits.
4. Determine the offset of the AND bits.
5. Create a device dependent bitmap with the XOR bits.
6. Obtain the device dependent XOR bitmask and save it in memory. (The

AND bitmask is monochrome. Monochrome bits are identical in both the device dependent bitmaps and device independent bitmaps so there is no need to convert the AND bitmask.)

7. Flip the monochrome AND bits by scanlines since a DIB is stored upside down.

8. Use the XIO and AND bits to create a cursor by calling **CreateCursor**.

See Also

NWAPPMakeIcon

NWAPPMakeIcon

Creates an icon based on the DIB information returned by **ReadIcon**

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

HICON NWAPPMakeIcon (
   HINSTANCE   ghInst,
   HANDLE      hDIB);
```

Parameters

ghInst

(IN) Specifies a handle to a module or application instance.

hDIB

(IN) Specifies a handle to the icon's DIB information.

Return Values

If successful, **NWAPPMakeIcon** returns a handle to a newly created icon. Otherwise, it returns NULL.

Remarks

The steps involved in creating an icon from a DIB are similar to the steps involved in creating a cursor from a DIB and include:

1. Obtain a pointer to the icon's DIB bits.
2. Divide the DIB'd height by 2 to account for the fact that the DIB stores both the XOR and the AND masks, one after the other.
3. Determine the offset of the XOR bits.
4. Determine the offset of the AND bits.
5. Create a device dependent bitmap with the XOR bits.
6. Obtain the device dependent XOR bitmask and save it in memory. (The AND bitmask is monochrome. Monochrome bits are identical in both the device dependent bitmaps and device independent bitmaps so there is no need to convert the AND bitmask.)
7. Flip the monochrome AND bits by scanlines since a DIB is stored

upside down.

8. Use the XIO and AND bits to create an icon by calling **CreateIcon**.

See Also

NWAPPMakeCursor

NWAPPModifySchema

Modifies the schema for application objects

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

BOOL N_API N_EXPORT NWAPPModifySchema (
   APP_CONTEXT_HANDLE   dContext);
```

Parameters

dContext

(IN) Specifies the context of the directory tree whose schema is to be modified.

Return Values

TRUE	Schema was successfully modified
FALSE	Schema could not be modified (it was already modified, the user does not have enough rights to modify it, or there is a conflict in classes or attributes)

Remarks

If the schema has already been partially modified, **NWAPPModifySchema** attempts to fully modify it.

NWAPPMonitorApplicationById

Checks to see if any applications launched by **NWAPPLaunchApplicationId** have terminated

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

nint16 N_API N_EXPORT NWAPPMonitorApplicationById (
   nint16   id);
```

Parameters

id

(IN) Specifies the unique ID of the application returned by calling **NWAPPLaunchApplicationId**.

Return Values

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

APP_ALIVE	
APP_KILLED	

Remarks

NWAPPMonitorApplicationById needs to be called periodically (preferably every 1-3 seconds).

If applications have terminated, **NWAPPMonitorApplicationById** performs appropriate cleanup for the terminated applications' resources.

NWAPPMonitorApplications

Checks to see if the specified application launched by **NWAPPLaunchApplication** has terminated

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

void N_API N_EXPORT NWAPPMonitorApplications (
   void);
```

Return Values

None

Remarks

NWAPPMonitorApplications needs to be called periodically (preferably every 1-3 seconds).

If applications have terminated, **NWAPPMonitorApplications** performs appropriate cleanup for the terminated applications' resources.

NWAPPProcessVariables

Substitutes words between percent signs to either user object attribute values or environment variables

Platform: Windows 3.1, Windows95

Service: Application Launcher

Syntax

```
#include <nwapp.h>

void N_API N_EXPORT NWAPPProcessVariables (
   APP_CONTEXT_HANDLE   context,
   pnstr8               str,
   nint16               strSize);
```

Parameters

context

(IN) Specifies the context you are working in.

str

(IN/OUT) Specifies the string to be modified on input. On output,*str* will be modified on output to contain the variable information.

strSize

(IN) Specifies the size, in bytes, of *str*.

Return Values

None

Remarks

NWAPPProcessVariables will substitute according to the following examples if username=djanis:

before calling NWAPPProcessVariables	**after calling NWAPPProcessVariables**
"\\server\vol\%username%"	"\\server\vol\djanis"
"\\server\vol\%dn%"	"\\server\vol\djanis"

"%dn%" becomes "djanis" because the DN of the user object is used.

before calling NWAPPProcessVariables	**after calling NWAPPProcessVariables**
“\\server\vol\%.djanis.cpt.npd.novell:email”	“\\server\vol\damon_janis@novell.com”

“novell:email” becomes “damon_janis@novell.com” because it uses the E-mail attribute of the .djanis.cpt.npd.novell object.

NOTE: If the string between the percent signs cannot be replaced or if there are no percent signs, *str* is returned untouched.

NWAPPReadIcon

Reads an icon resource file and extracts the DIB information
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

HANDLE NWAPPReadIcon (
   pnstr8    iconData,
   nuint32   cbIconData);
```

Parameters

iconData

(IN) Points to the icon resource file to read.

cbIconData

(IN) Specifies the number of bytes to which *iconData* points.

Return Values

If successful, **NWAPPReadIcon** returns a handle to a DIB. Otherwise, it returns NULL.

Remarks

Only the first icon in a given file is read. To modify **NWAPPReadIcon** to read in all the icons in an .ICO file, you must read in the entire icon resource directory and identify the bits to each icon.

See Also

NWAPPGetIconFromHandle

NWAPPStripFullObjectName

Strips the typed name of an NDS object to the object name
Platform: Windows 3.1, Windows95
Service: Application Launcher

Syntax

```
#include <nwapp.h>

pnstr8 N_API N_EXPORT NWAPPStripFullObjectName (
   APP_CONTEXT_HANDLE   context,
   pnstr8               fullName,
   pnstr8              strippedName);
```

Parameters

context

(IN) Specifies the context of the object.

fullName

(IN) Points to the typed name of the object.

strippedName

(OUT) Points to the stripped name of the object. You must allocate memory to contain this name.

Return Values

If successful, **NWAPPStripFullObjectName** returns a pointer to the stripped name.

Remarks

NWAPPStripFullObjectName strips the typed name of an object down to the object name. For example, `CN=DJanis.O=Novell` becomes `DJanis`.

Application Launcher: Structures

APP_APPLICATION_OBJECT

Defines an application object and values for its attributes

Service: Application Launcher

Defined In: nwapp.h

Structure

```
typedef struct
{
   nstr8          className[MAX_SCHEMA_NAME_BYTES];
   nstr8          name[MAX_DN_BYTES];
   nstr8          description[MAX_DESCRIPTION_LENGTH];
   nstr8          path[MAX_PATH_LENGTH];
   nstr8          workingDirectory[MAX_PATH_LENGTH];
   nstr8          commandLineParameters[MAX_PARAMETERS_LENGTH];
   nstr8          longDescription[MAX_BLURB_LENGTH];
   uint32         flags;
   nstr8          driveMappings[3][MAX_MISC_LENGTH];
   nint16         driveCount;
   nstr8          printerPorts[3][MAX_MISC_LENGTH];
   nint16         printerCount;
   NWFILE_HANDLE  preLaunchScript;
   NWFILE_HANDLE  postLaunchScript;
   APP_NAME_LIST  *contacts;
   nint16         contactcount;
   APP_NAME_LIST  *associations;
   nint16         associationCount;
} APP_APPLICATION_OBJECT;
```

Fields

className

Specifies the class name. For example:

C_WINDOWS_APPLICATION
C_WIN_APPLICATION
C_WIN_NT_APPLICATION

name

Specifies the object name

description

Specifies the application description. For example, “GroupWise 3.0”.

path

Specifies the path where the executable is located (including the executable name). For example:

UNC path: \\>server name<\>volume<\>dir path<\>dir path<\grpwise.exe

Drive based: F:\...\grpwise.exe

workingDirectory

Specifies the working directory for the executable---UNC or Drive based (optional).

commandLineParameters

Specifies the command line parameters to be passed to the executable (optional).

longDescription

Specifies the long description of the application (optional).

flags

Specifies the flags to set (optional). For example: APP_FLAG_MINIMIZE | APP_FLAG_NO_CLEANUP. Set to zero for none.

driveMappings

Specifies the map commands for up to 3 drive mappings (optional). For example:

driveMappings[0] = “F:=\\server\volume”

driveMappings[1] = “N=\\server\volume”

driveCount

Specifies the number of drives defined by *driveMappings*. Set to zero for none.

printerPorts

Specifies the capture commands for up to 3 ports (optional). For example:

printerPorts[0] = “LPT1:=\\server\queue”

printerPorts[1] = “LPT2:=\\server\queue”

printerCount

Specifies the number of ports defined in *printerPorts*. Set to zero for none.

preLaunchScript

Specifies the open file handle to an ASCII text file containing a pre-launch script. Set to NULL for none.

postLaunchScript

Specifies the open file handle to an ASCII text file containing a post-launch script. Set to NULL for none.

contacts

Points to an array of contacts (users in the DS tree) that are contacts for this application.

contactCount

Specifies the number of contacts in the array pointed to by *contacts*. Set to zero for none.

associations

Points to an array of associations (users, groups, or containers that can set this application). Set to zero for none.

associationCount

Specifies the number of associations in *associations*. Set to zero for none.

Remarks

name cannot contain periods and other string characters disallowed by DS object names.

preLaunchScript and *postLaunchScript* should contain an ASCII text file containing login-script syntax.

NOTE: The NWAPP library will not close the file.

You must allocate and free memory for the array pointed to by *contacts*. For example, to add 3 contacts:

```
APP_APPLICATION_OBJECT *appData;
appData->contactCount = 3;
appData->contacts = (APP_NAME_LIST *)malloc(3 * sizeof(APP_NAME_LIST));
strcpy(appData->contacts[0], ".cn-djanis.ou=cpt.ou=npd.o=novell");
strcpy(appData->contacts[1], ".dn-carlah.ou=cpt.ou=npd.o=novell");
strcpy(appData->contacts[2], ".cn=mgbrooks.ou=cpt.ou=npd.o=novell");
...
free(appData->contacts);
```

You must allocate and free memory for the array pointed to by *associations*. For example, to add 3 associations:

```
APP_APPLICATION_OBJECT *appData;
appData->associationCount = 3;
appData->associations = (APP_NAME_LIST *)malloc(3 * sizeof(APP_NAME_LIS
strcpy(appData->associations[0], ".cn=djanis.ou=cpt.ou=npd.o=novell");
strcpy(appData->associations[1], ".ou=cpt.ou=npd.o=novell");
strcpy(appData->associations[2], ".cn=CPT_GROUP.ou=cpt.ou=npd.o=novell"
...
free(appData->associations);
```

NOTE: The fully qualified DS name needs to be in the *contacts* and *associations* array. The contact list is of type SYN_DIST_NAME (distinguished name) which requires a real DS object.

APP_ATTRIBUTE_VALUE

Defines an attribute value

Service: Application Launcher

Defined In: nwapp.h

Structure

```
typedef struct AttributeValue
{
   NWSYNTAX_ID              syntaxID;
   nptr                     value;
   uint32                   valueSize;
   struct AttributeValue   *next;
} APP_ATTRIBUTE_VALUE;
```

Fields

syntaxID

Specifies the syntax of the attribute.

If the *syntaxID* is APP_STREAM_ATTRIBUTE, the value points to an open file handle (HFILE in Windows). This file is closed when the APP_ATTRIBUTE_VALUE is freed by calling **NWAPPFreeGetObjectAttributes**. Use this file handle as you would any other.

value

Points to the value of the attribute.

valueSize

Specifies the size in bytes of the value.

next

Points to the next attribute value in the list.

APP_DESKTOP_FOLDER

Defines a generic desktop group
Service: Application Launcher
Defined In: nwapp.h

Structure

```
typedef struct DesktopFolder
{
   pnstr8                 name;
   APP_DIRECTORY_OBJECT  *head;
   struct DesktopFolder  *next;
} APP_DESKTOP_FOLDER;
```

Fields

name

Specifies the name of the desktop group.

head

Points to the list of objects in this folder.

next

Points to the next desktop object in the list.

APP_DIRECTORY_OBJECT

Defines an NDS object
Service: Application Launcher
Defined In: nwapp.h

Structure

```
typedef struct DirectoryObject
{
   uint32                   type;
   pnstr8                   name;
   pnstr8                   fullName;
   struct DirectoryObject  *next;
} APP_DIRECTORY_OBJECT;
```

Fields

type

Specifies the type of application:

2 APP_AUTO_START
4 APP_DESKTOP

name

Specifies the short name of the object (that is, the object name relative to the current context).

fullName

Specifies the fully-qualified name of the object.

next

Points to the next NDS object in the list.

APP_OBJECT_ATTRIBUTE

Defines a generic attribute for an NDS object
Service: Application Launcher
Defined In: nwapp.h

Structure

```
typedef struct ObjectAttribute
{
   pnstr8                      name;
   APP_ATTRIBUTE_VALUE        *valueHead;
   struct ObjectAttribute     *next;
} APP_OBJECT_ATTRIBUTE;
```

Fields

name

Specifies the name of the attribute.

valueHead

Points to the first node in the value list, or N_NULL if there are no values for the attribute.

next

Points to the next attribute in the list.

Auditing

Auditing: Guides

Auditing: General Guide

Auditing: Task Guide

Auditing: Concept Guide

Auditing: Functions

Auditing: Structures

Parent Topic:

Auditing: General Guide

The Audit File Configuration Header

This topic discusses the audit file configuration header and how it controls the auditing on the associated volume or container.

The Audit File Configuration Header Introduction

The Audit File Configuration Header Example

Auditing Flags

Auditing Flags Example

The Audit File Configuration Header Event Bitmap

Scope of Auditing Events

Auditing File Events (NetWare 4.x)

Reading a Volume Event Bitmap

Event Bits Tables

Parent Topic:

Auditing: General Guide

Auditing Event Records

Event records are generated according to the event bitmap in the configuration header. These records make up the data segment of the audit file. The configuration header includes an *audit record count* that records the current number of event records.

Fields of Auditing Event Records

Reading Auditing Event Records

Parent Topic:

Auditing: General Guide

Auditing File Events (NetWare 4.x)

For NetWare 4 versions prior to 4.11, file events are the only category of events for which you can target specific files and users. To audit file events you must configure the files and the user objects in addition to setting the file event bit in the volume's event bitmap. To target specific files you must set each file's audit attribute. To target specific users you must add an audit property to each user's bindery object. Both procedures are explained below.

User Audit Property

File Events for Auditing

Parent Topic:

The Audit File Configuration Header Event Bitmap

Enabling and Disabling Auditing

Enabling and disabling auditing is different depending on whether you are using NetWare 4.11 or previous versions of NetWare 4. However, in both cases, when auditing is enabled an audit file is automatically created. This file is where the audit data is stored.

Enabling and Disabling Auditing for NetWare 4 Versions Prior to 4.11

Enabling Auditing for NetWare 4 Versions Prior to 4.11

Disabling Auditing for NetWare 4 Versions Prior to 4.11

Auditing Passwords for NetWare 4 Versions Prior to 4.11

Enabling and Disabling Auditing for NetWare 4.11

Enabling Auditing for NetWare 4.11

Disabling Auditing for NetWare 4.11

AFO Attributes for NetWare 4.11

Parent Topic:

Auditing: General Guide

Event Bits Tables

This is a list of tables that define event bits.

Accounting Events Table

File Events

Message Events Table

QMS Events Table

Server Events Table

User Events Table

Directory Services Events Table

Additional Directory Services Events for NetWare 4.11 Table

Parent Topic:

The Audit File Configuration Header Event Bitmap

Reading a Volume Event Bitmap

The volume event bitmap is a bit stream 512 bits long. Special functions are defined to simplify access to this bitmap:

NWADReadBitMap reads the event bitmap.

NWADWriteBitMap modifies the event bitmap.

Reading a Volume Event Bitmap Example

Parent Topic:

The Audit File Configuration Header Event Bitmap

Auditing: Tasks

Disabling Auditing for NetWare 4 Versions Prior to 4.11

1. **Call NWADDisable to disable auditing and close the audit file.**

Parent Topic:

Enabling and Disabling Auditing

Related Topics:

Enabling Auditing for NetWare 4 Versions Prior to 4.11

Auditing Passwords for NetWare 4 Versions Prior to 4.11

Disabling Auditing for NetWare 4.11

1. **Call NWADClose to free the audit handle allocated by NWADOpen.**

Parent Topic:

Enabling and Disabling Auditing

Related Topics:

Enabling Auditing for NetWare 4.11

AFO Attributes for NetWare 4.11

Enabling Auditing for NetWare 4 Versions Prior to 4.11

1. **Call NWADLogin to initialize the auditing password.**
2. **Call NWADEnable to enable auditing on the object.**

   NOTE: For first time auditing, the login function returns ERR_AUDITING_NOT_ENABLED. You can then call **NWADEnable** to enable auditing. The first-time auditor must have supervisor equivalence.

   Don't confuse logging in to a volume or container as auditor with logging in to a NetWare® server. Audit login is a unique procedure implemented through a special service request.

If a second-level password is in effect, it must also be initialized. The auditor password is required even when auditing is disabled. For more information see Auditing Passwords for NetWare 4 Versions Prior to 4.11.

When you log in to an object, Auditing returns an audit key. The key permits access to the audit data for the object you have logged in to by providing access to the password calls. Remember to remove the password from memory as soon as you receive the audit key. The audit key is the basis for security throughout an auditing session.

Parent Topic:

Enabling and Disabling Auditing

Related Topics:

Disabling Auditing for NetWare 4 Versions Prior to 4.11

Auditing Passwords for NetWare 4 Versions Prior to 4.11

Enabling Auditing for NetWare 4.11

1. **Create a DS Audit File Object (AFO) in the container you want to audit using either DS calls or the Auditcon utility.**

2. **Give a user (the auditor) read and write rights to the Audit:Policy and Audit:Contents attributes of the AFO.**

3. **Call NWADOpen to allocate an audit handle for use in other auditing functions.**

   NOTE: The creator of the AFO is automatically given auditor status. Remove the creator's DS rights to the AFO if you do not want the creator to have auditor access.

Parent Topic:

Enabling and Disabling Auditing

Related Topics:

Disabling Auditing for NetWare 4.11

AFO Attributes for NetWare 4.11

Auditing: Examples

The Audit File Configuration Header Example

The following code writes some arbitrary values to a container's configuration header. Since all data items in the header are affected by the operation, the header is read first to obtain values for those items that will not be modified.

Writing to a Container's Configuration Header

```
/* ******************************************************************
 *
 * Name        : Writing to a container's configuration header
 *
 *
 * Abstract    : Writes arbitrary values to a container's configuration
 *               Since all data items in the header are affected by the
 *               operation, the header is first read using
 *               NWADReadConfigHeader to obtain values for those items
 *               will not be modified.
 *
 * ******************************************************************

#ifdef N_PLAT_OS216
#include <os2.h>
#endif

#ifdef N_PLAT_WIN16
#include <windows.h>
#endif

#include <stdio.h>
#include <time.h>
#include <nwaudit.h>
#include <nwdsaud.h>
#include <nwmisc.h>

void main(void)
{
nuint32              auditIDType=1; /* Set to Container for this exampl
nuint32              auditID = 0;
pNWADOpenStatus      openStatus;
pnptr                auditHandle;
NWCONN_HANDLE        connHandle;
NWRCODE              ccode = 0;
```

```
nchar                ObjectName[40];  /* ie. .OU=MyDept.O=MyCompany */
NWConfigHeader       buffer;
nuint16              bufferSize;

NWDSContextHandle    contextHandle;

   buffer.errMsgDelayMinutes=0;
   buffer.auditFileMaxSize=0;
   buffer.auditFileSizeThreshold=0;


/* Assume values have been assigned to the connection handle and contex


   ccode = NWCallsInit(NULL,NULL);

   if (auditIDType==1) /* Type 0 is Volume Auditing, 1 is Container Aud

   {
       ccode = NWDSAuditGetObjectID(contextHandle,

                                     objectName,

                                     &connHandle,
                                &auditID);
   }

   ccode = NWADOpen(connHandle,
                    auditIDType,
                    auditID,
                    auditHandle,
                    openStatus);
   if (ccode)
   {
         printf("\nUse AUDITCON to enable auditing on server");
             /*return(-1);      */
   }

   bufferSize = sizeof(NWConfigHeader);
   ccode = NWADReadConfigHeader(connHandle,

                                 auditIDType,
                                  auditID,

                                 &buffer,

                                 bufferSize);
   buffer.errMsgDelayMinutes+=10;
   buffer.auditFileMaxSize+=50;
   buffer.auditFileSizeThreshold+=60;
   ccode = NWADWriteConfigHeader(connHandle, auditIDType,
  (nuint32)auditID, au
ditHandle, );
   ccode = NWADLogout(connHandle,

                      auditIDType,

                      auditID,

                      auditHandle);
   ccode = NWADClose(auditHandle );
}
```

Parent Topic:

The Audit File Configuration Header Introduction

Auditing Flags Example

The following code calls **NWADGetFlags** and uses constants defined in nwaudit.h to read the auditing flags on a volume.

Reading Auditing Flags on a Volume

```
/* *******************************************************************
 *
 * Name        : Reading the auditing flags on a volume
 *
 *
 * Abstract    : Call NWADGetFlags and use the constants defined in
 *               nwaudit.h to read the auditing flags on a volume.
 *
 * Inputs      : VOID
 *
 * Notes       : This example shows relevant audit APIs only and assume
 *               values have been assigned to connection handle, vol nu
 *               and audit key variables.
 * *******************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <nwnet.h>
#include <nwaudit.h>

void main(void)
{
     nuint32            auditIDType;
     NWCONN_HANDLE      conn;
     nuint8             flags;
     NWCCODE            ccode;
     nuint32            auditID;
     nptr               auditHandle;
/*   Get the flags for the given volume */
     ccode = NWADGetFlags(conn,

                          auditIDType,
                          auditID,

                          auditHandle,

                          &flags);

     if(ccode == 0)
     {
/*      Print the audit information */
        printf("\nThe following Auditing Flags are set:\n");
        if(flags + DiscardAuditRcdsOnErrorFlag)
           printf("\n DiscardAuditRcdsOnErrorFlag 0x01 is set.");
        if(flags + ConcurrentVolAuditorAccess)
           printf("\n ConcurrentVolAuditorAccess 0x02 is set.");
        if(flags + DualLevelPasswordsActive)
           printf("\n DualLevelPasswordsActive 0x04 is set.");
        if(flags + BroadcastWarningsToAllUsers)
           printf("\n BroadcastWarningsToAllUsers 0x08 is set.");
        if(flags + LevelTwoPasswordSet)
           printf("\n LevelTwoPasswordSet 0x10 is set.");
    }
}
```

Parent Topic:

Auditing Flags

Reading a Volume Event Bitmap Example

The following code calls **NWADReadBitMap** to read the event bitmap.

Reading the Event Bitmap

```
/* ***********************************************************************
 *
 * Name         : Reading the event bitmap
 *
 *
 * Abstract     : Call NWADReadBitMap to read the event bitmap
 *
 * Notes        : This example shows relevant audit APIs only and assume
 *                values have been assigned to connection handle, vol nu
 *                and audit key variables.
 * ***********************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <nwnet.h>
#include <nwaudit.h>

void main(void)
{
     NWAuditBitMap      buffer;
     NWCONN_HANDLE      conn;
     NWCCODE            ccode;
     nuint32            auditID;
     nuint16            bufferSize
     int                i, j, flag;

/*   Assume values have been assigned to the connection handle and audi

/*   Read the event bitmap for the specified volume  */



     bufferSize = sizeof(buffer);

     ccode = NWADReadBitMap(conn,

                           auditID,
                           &buffer,
                           bufferSize);
     if (ccode)
     {
          printf("\nNWADReadBitMap returned %x\n", ccode);
          exit;
     }
     printf("\n Bit map: " );
     for(i = 0; i < 10; i++)
     {
        for(j = 0, flag = 1; j < 8; j++)
        {
                printf(" %c",(flag & buffer.bitMap[i]) ? '1' : '0');
                flag = flag << 1;
                /* there are only about 75 bits so break out after abou
                   many get printed  */
                if(i == 9 && j == 3)
                        break;
        }
        if(i == 9) break;
        else
           printf("\n   :");
     }
}
```

Parent Topic:

Reading a Volume Event Bitmap

Auditing: Concepts

Accounting Events Table

The following table lists the bits defined in the accounting events bitmap.

Table auto. Accounting Events

Event Type ID	Event ID
A_EVENT_GET_CURRET_ACNT_STATS	200
A_EVENT_SUBMIT_ACCOUNT_CHARGE	201
A_EVENT_SUBMIT_ACCOUNT_HOLD	202
A_EVENT_SUBMIT_ACCOUNT_NOTE	203

Parent Topic:

Event Bits Tables

Additional Directory Services Events for NetWare 4.11 Table

ADS_ADD_MEMBER	129	29	unicode ObjectName LONG MemberID unicode PropertyName
ADS_BACKUP_ENTRY	130	30	unicode EntryName
ADS_CHANGE_BIND_OBJ_SECURITY	131	31	unicode ObjectName LONG NewObjectSecurity
ADS_CHANGE_PROP_SECURITY	132	32	unicode propertyName LONG newPropertySecurity

ADS_CHANGE_TREE_NAME	133	33	unicode NewTreeName
ADS_CHECK_CONSOLE_OPERATOR	134	34	unicode ServerName LONG OperatorPrivGranted
ADS_COMPARE_ATTR_VALUE	135	35	unicode attrName
ADS_CREATE_PROPERTY	136	36	unicode ObjectName unicode PropertyName LONG PropertySecurity
ADS_CREATE_SUBORDINATE_REF	137	37	unicode EntryName
ADS_DEFINE_ATTR_DEF	138	38	unicode attrName
ADS_DEFINE_CLASS_DEF	139	39	unicode className
ADS_DELETE_MEMBER	140	40	unicode ObjectName LONG MemberID unicode PropertyName
ADS_DELETE_PROPERTY	141	41	unicode ObjectName unicode MemberID
ADS_DS_NCP_RELOAD	142	42	
ADS_RESET_DS_COUNTERS	143	43	unicode ClientServerName
ADS_FRAG_REQUEST	144	44	
ADS_INSPECT_ENTRY	145	45	unicode EntryName
ADS_LIST_CONTAINABLE_CLASSES	146	46	unicode EntryName
ADS_LIST_PARTITIONS	147	47	unicode PartitionRootName
ADS_LIST_SUBORDINATES	148	48	unicode EntryName
ADS_MERGE_ENTRIES	170	70	unicode winnerEntry unicode loserEntry
ADS_MERGE_TREE	149	49	
ADS_MODIFY_CLASS_DEF	150	50	unicode className
ADS_MOVE_TREE	151	51	unicode EntryName
ADS_OPEN_STREAM	152	52	unicode EntryName unicode attrName unicode FileName LONG desiredRights
ADS_READ	153	53	unicode EntryName
ADS_READ_REFERENCES	154	54	unicode EntryName

ADS_REMOVE_ATTR_DEF	155	55	unicode attrName
ADS_REMOVE_CLASS_DEF	156	56	unicode className
ADS_REMOVE_ENTRY_DIR	157	57	unicode EntryName
ADS_RESTORE_ENTRY	158	58	unicode EntryName
ADS_START_JOIN	159	59	unicode ParentRootEntryName unicode ChildRootEntryName
ADS_START_UPDATE_REPLICA	160	60	unicode replicaName
ADS_START_UPDATE_SCHEMA	161	61	unicode ClientServerName
ADS_SYNC_PARTITION	162	62	unicode partitionDistName
ADS_SYNC_SCHEMA	163	63	
ADS_UPDATE_REPLICA	164	64	unicode replicaName
ADS_UPDATE_SCHEMA	165	65	unicode ClientServerName
ADS_VERIFY_PASSWORD	166	66	unicode EntryName
ADS_ABORT_JOIN	167	67	
ADS_LAST_PLUS_ONE	168	68	

Parent Topic:

Event Bits Tables

AFO Attributes for NetWare 4.11

By creating the AFO using the mandatory DS attributes you give the user auditor status and access to the audit file. The AFO has two mandatory and several optional NDS attributes. The mandatory attributes are as follows:

Audit:Contents is where read and write rights are assigned to the auditor.

Audit:Policy is where you define the audited events, audit archiving options, audit file overflow recovery options, maximum size of the audit file, whether encryption is used, and the number of old audit files.

The optional attributes are as follows:

Audit:A Encryption Key

Audit:B Encryption Key

Audit:Current Encryption Key

Audit:Path points to the path to the NDS volume where the audit file resides.

Audit:Link List points to the container, volume, or one or more workstations for which the AFO stores audit

Audit Type specifies the type of auditing you are doing:

0 = volume auditing
1 = container auditing
2 = external auditing

Also included are all the standard minimum attributes inherited from Super Class Top.

Parent Topic:

Enabling and Disabling Auditing

Related Topics:

Enabling Auditing for NetWare 4.11

Disabling Auditing for NetWare 4.11

The Audit File Configuration Header Event Bitmap

The event bitmap in the configuration header is the key to configuring the audit file. Each bit in the event bitmap represents an event to be audited. In its initial state, none of the bits in the event bitmap is set. It's up to the auditor to access the audit file's configuration header and set the bits that correspond to events the auditor wants audited.

The event bitmap for containers is a 32-bit value with 32 corresponding events. The event bitmap for volumes is 512 bits (64 bytes).

Events that can be audited fall into several categories:

Directory Services Events

Bindery Events

NetWare® Server Events

Queue Management Events

File System Events

User Object Events

You must use container auditing to audit Directory Services events. All other events must be audited on a volume basis. For each category there are many events you can audit. Setting the bit for an event results in an event record being added to the audit file every time that event occurs.

Parent Topic:

The Audit File Configuration Header

Related Topics:

Scope of Auditing Events

Auditing File Events (NetWare 4.x)

Reading a Volume Event Bitmap

Event Bits Tables

The Audit File Configuration Header Introduction

The audit file configuration header controls the auditing on the associated volume or container. Although configuration headers differ somewhat between volumes and containers, both contain the following key fields:

A set of audit flags that indicate the status of auditing for the object.

An event bitmap that determines which events are audited.

A count of the current number of events that have been recorded in the audit file.

Additional fields contain information about the state of the audit file. The structure NWConfigHeader contains the volume configuration header data. The structure NWDSContainerConfigHdr contains the container configuration header data.

NWADAuditPolicy is where Auditing stores the policy data.

The configuration header for NetWare® 4.11 is twice the size of that for previous versions of NetWare. This allows for more events to be audited.

Use the following functions to read from or write to the configuration headers:

NWADReadConfigHeader reads an object header.

NWADWriteConfigHeader modifies an object header.

In the case of volumes, you can also read or write just the event bitmap in the header. For more information see The Audit File Configuration Header Event Bitmap.

Read or write operations affect the entire configuration header. When modifying a header, be sure to assign values to all the header's data items.

Parent Topic:

The Audit File Configuration Header

Related Topics:

The Audit File Configuration Header Example

Audit File Functions

These functions open and close the audit file and the history file and read event records from these files.

Function	Header	Comment
NWADAppendExternalRecords	nwaudit.h	Adds external audit events to an event record.
NWADCloseOldFile	nwaudit.h	Closes the audit file that was opened when auditing was enabled.
NWADCloseRecordFile	nwaudit.h	Frees *recordHandle* allocated by **NWADOpenRecordFile**.
NWADDeleteFile	nwaudit.h	Deletes the old file on the specified object.
NWADDeleteOldFile	nwaudit.h	Deletes an old file in the file list.
NWADGetFileList	nwaudit.h	Returns the file list.
NWADOpenRecordFile	nwaudit.h	Opens the record file and allocates a *recordHandle*.
NWADResetFile	nwaudit.h	Closes the object's current auditing file and opens a new one.
NWADReadRecord	nwaudit.h	Reads a specified record.

Parent Topic:

Auditing: General Guide

Audit History Files

The audit file is automatically created when auditing is enabled and is accessible only to the auditor.

To close the current audit file and open a new one call **NWADResetFile**. The closed file becomes a history file. Up to15 audit history files can be maintained by Auditing.

Several functions allow you to manipulate the audit files.

NWADDeleteFile deletes an old audit file.

NWADOpenRecordFile opens a record file and allocates a *recordHandle*.

NWADCloseRecordFile closes the record file and frees the *recordHandle* allocated by **NWADOpenRecordFile**.

NWADReadRecord reads a specified record.

NWADGetFileList returns the list of files.

A parameter in some of these functions allows you to specify which file you want to use. Set the parameter to -1 to view the current file. To view history files set the parameter to a number from 0 to 15.

The audit file consists of a configuration header and a data segment. The configuration header specifies what type of events are to be audited. The data segment stores event records describing events that have occurred. For more information see The Audit File Configuration Header.

IMPORTANT: Be sure to close and remove all audit files before deleting a container on which auditing has been enabled.

Parent Topic:

Auditing: General Guide

Audit Property Functions

These functions add, remove, and check for an object's audit property. By adding the property to a user object, you can audit the user's file operations individually.

Function	**Header**	**Comment**
NWADChangeObjectProperty	nwaudit.h	Changes the audit property for a user on the specified object.
NWGetNWADVersion	nwaudit.h	Returns version information about the NWAD library.
NWADIsObjectAudited	nwaudit.h	Determines whether the specified object or user is being audited.

Parent Topic:

Auditing: General Guide

Audit Security Functions

These functions control auditor access to volumes and containers (objects) and perform operations on auditor passwords.

Function	Header	Comment
NWADChangePassword	nwaudit.h	Changes the auditor's password for the specified object.
NWADCheckAccess	nwaudit.h	Determines whether the caller has level 1 auditor access to the object.
NWADCheckLevelTwoAccess	nwaudit.h	Determines whether the caller has level 2 auditor access to the object.
NWADInitLevelTwoPassword	nwaudit.h	Initializes an audit key for level 2 access.
NWADLogin	nwaudit.h	Initializes an audit key for level 1 access to the object.
NWADLogout	nwaudit.h	Removes the caller's auditor status from an object.
NWADOpen	nwaudit.h	Allocates *auditHandle* for use in other Auditing functions.
NWADRestartVolumeAuditing	nwaudit.h	Restarts auditing for volumes only.
NWADSetPassword	nwaudit.h	Sets a password on for NetWare® 4.11 only.

Parent Topic:

Auditing: General Guide

Audit Status Functions

These functions enable and disable auditing on volumes and containers (objects) and return audit status information.

Function	Header	Comment
NWADDisable	nwaudit.h	Disables auditing on the specified object.
NWADEnable	nwaudit.h	Enables auditing on the specified object.
NWADGetFlags	nwaudit.h	Returns the auditing flags from the object's audit file configuration header.
NWADGetStatus	nwaudit.h	Returns auditing status information for the specified object.

Parent Topic:

Auditing: General Guide

Audit Status Information

Audit status information indicates whether auditing is enabled on a specified object. If auditing is enabled, the information also includes statistics on the object's audit file. This information is available to any client that has logged in unless you are using NetWare® 4.11, in which case very little information is available to any client.

NWADGetStatus returns audit status information for objects.

Audit status information is returned as an NWAuditStatus structure, indicating the size and version of the audit file.

Parent Topic:

Auditing: General Guide

Auditing Flags

Auditing flags can be read separately from other data in the audit file configuration header. The flags control several important aspects of the auditing process and are stored as a set of bit flags in a single-byte value. There are five flags:

Byte 0 = DiscardAuditRcdsOnErrorFlag
Byte 1 = ConcurrentVolAuditorAccess
Byte 2 = DualLevelPasswordsActive (this byte is ignored with NetWare® 4.11)
Byte 3 = BroadcastWarningsToAllUsers
Byte 4 = LevelTwoPasswordSet (this byte is ignored with NetWare 4.11)

If DiscardAuditRcdsOnErrorFlag is set, the auditing records are discarded if an error occurs.

If ConcurrentVolAuditorAccess is set more than one auditor can log into the object at a time.

If DualLevelPasswordsActive is set, dual level password security is in effect.

If BroadcastWarningsToAllUsers is set, audit warnings are broadcast to all users.

If LevelTwoPasswordSet is set, the level 2 password has been assigned a value. (This last flag is meaningful only if the dual level passwords flag is set.) Bits 0 through 3 can be set by the auditor.

NWADGetFlags returns the auditing flags for an object.

Parent Topic:

The Audit File Configuration Header

Related Topics:

Auditing Flags Example

Auditing Introduction

NetWare® auditing allows NetWare 4.x servers to generate files that track and record network events. Audit files assist independent auditors in the routine auditing of the network environment. An auditor can enable auditing on a volume or a container, set up an audit for specific events, and read the record of events that have occurred.

Auditing can also be applied to other activities. For example, an administrator might set up an audit file as a log for analyzing and monitoring network usage, or an NLM™ application might use an audit file to track special events. Auditing provides the interface to the auditing system, giving your applications secure procedures for setting up audits and reading the results of auditing sessions.

For a description of structures and other data definitions relating to this, see Auditing: Structures.

Parent Topic:

Auditing: General Guide

Auditing Passwords for NetWare 4 Versions Prior to 4.11

Access control to an audit file can be single-tiered or two-tiered. Two-tiered security uses two passwords: one for reading audits and one for configuring audits. As an auditor, you activate two-tiered security for an audited object by calling **NWADInitLevelTwoPassword**.

The current level of audit security is indicated by an object's auditing flags. For more information, see Auditing Flags. Two functions verify an auditor's level 1 and level 2 access:

NWADCheckAccess checks level 1 access.

NWADCheckLevelTwoAccess checks level 2 access.

To change either the level 1 or level 2 password, you must be logged in to the object as auditor and receive a valid audit key. Call **NWADChangePassword** to change the passwords.

Parent Topic:

Enabling and Disabling Auditing

Related Topics:

Enabling Auditing for NetWare 4 Versions Prior to 4.11

Disabling Auditing for NetWare 4 Versions Prior to 4.11

Auditing Security

Audit security is different depending on whether you are using NetWare® 4.11 or previous versions of NetWare 4.x. Previous versions of NetWare 4.x use passwords to access and configure the audit files. The network admin enables auditing and assigns an initial password to the auditor. However, the admin isn't the security equivalent of the auditor and consequently, changing the auditor's password can prevent even the admin from accessing the audit file.

Access to the audit file with NetWare 4.11 is based on an NDS object, the Audit File Object (AFO), and an auditor's read and write rights to the object. The network admin creates the AFO and assigns rights to the auditor.

Parent Topic:

Auditing: General Guide

Auditing Volumes and Containers

You can set up auditing for either volumes or Directory Services (DS) container objects. Use volume auditing to audit the following:

Accounting events
Extended Attribute events

File events
Message events
QMS events
Server events
User events

Container auditing allows you to audit container DS events.

When you enable auditing on a container, you audit all objects directly subordinate to the container. You can audit as many objects as you choose by selecting the appropriate containers on which to enable auditing.

When you enable auditing on a volume, you can audit all objects located on the volume

In this document, when there are no differences whether you are auditing a volume or a container, the term object is used to refer to either a volume or container.

Parent Topic:

Auditing: General Guide

C2 Auditing Security Requirements

Security is an important aspect of Auditing due to the sensitive nature of auditing procedures and records. NetWare® 4.11 auditing is designed to meet the Class C2 security criteria as specified by the US Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC) also known as the "orange book."

Parent Topic:

Auditing: General Guide

Configuration Header Functions

These functions read and write the data stored in the audit file configuration header.

Function	Header	Comment
NWADReadConfigHeader	nwaudit.h	Returns the audit file configuration header for the specified object.
NWADWriteConfigHeader	nwaudit.h	Saves the configuration header data to the audit file on the specified object.

NWADReadBitMap	nwaudit.h	Returns the event bitmap in the audit file configuration header for the specified volume.
NWADWriteBitMap	nwaudit.h	Writes values to the event bitmap in the audit file configuration header for the specified volume.

Parent Topic:

Auditing: General Guide

Directory Services Events Table

The following table lists the bits defined in the DS event bitmap.

Table auto. Directory Services Events

Event Type ID	**Event ID**
ADS_ADD_ENTRY	101
ADS_REMOVE_ENTRY	102
ADS_RENAME_OBJECT	103
ADS_MOVE_ENTRY	104
ADS_CHANGE_SECURITY_EQUIV	105
ADS_CHG_SECURITY_ALSO_EQUAL	106
ADS_CHANGE_ACL	107
ADS_CHG_STATION_RESTRICTION	108
ADS_LOGIN	109
ADS_LOGOUT	110
ADS_CHANGE_PASSWORD	111
ADS_USER_LOCKED	112
ADS_USER_UNLOCKED	113
ADS_USER_DISABLE	114
ADS_USER_ENABLE	115
ADS_CHANGE_INTRUDER_DETECT	116
ADS_ADD_PARTITION	117
ADS_REMOVE_PARTITION	118
ADS_ADD_REPLICA	119

ADS_REMOVE_REPLICA	120
ADS_SPLIT_PARTITION	121
ADS_JOIN_PARTITION	122
ADS_CHANGE_REPLICA_TYPE	123
ADS_REPAIR_TIME_STAMPS	124
ADS_MOVE_SUB_TREE	125
ADS_ABORT_PARTITION_OP	126
ADS_SEND_REPLICA_UPDATES	127
ADS_RECEIVE_REPLICA_UPDATES	128

Parent Topic:

Event Bits Tables

Fields of Auditing Event Records

Although the information in an event record varies for volumes and containers, the key fields in both headers are *process unique ID* and *event type ID*. The process unique ID is the bindery object ID identifying the object that performed the event. The event type ID identifies the type of event audited.

Any event-specific information is appended to the end of the record as an event field. By checking the event type ID, you can determine how to interpret the event. For example, if the event type ID is A_EVENT_BIND_CREATE_PROPERTY is returned, a bindery property was created, and the event field contains the property name and security value.

The event records also record the date and time of the event and its success or failure. Volume events record the connection ID of the station where the event was performed. Container events record the user ID of the object performing the event. Container events also record the replica an event was performed on.

The NWVolAuditRecord structure contains the event record information for an event on a volume. The Account structure contains the event record information for an event on a container.

Parent Topic:

Auditing Event Records

Related Topics:

Reading Auditing Event Records

File Events for Auditing

Once you set the audit attribute for specific files or add the audit property to specific users, the next step is to set the event bitmap for file operations you want to audit. Some file events are represented by three different bits in the event bitmap: a global bit, a union bit, and an intersection bit. (See theFile Events.) Set the one appropriate for the events you are auditing.

Open File audit is an example of the choices presented by the three bits:

The Global Open File bit generates an event record every time a file is opened (regardless of which file is involved or who opens it).

The Union Open File bit generates an event record whenever an audited file is opened or an audited user opens a file.

The Intersection Open File bit generates an event record only when an audited user opens an audited file; other open file operations are ignored.

Parent Topic:

Auditing File Events (NetWare 4.x)

Related Topics:

User Audit Property

File Events

The following table lists the bits defined in the File event bitmap.

Event Type ID	**Event ID**
A_EVENT_CREATE_DIRECTORY	75
A_EVENT_DELETE_DIRECTORY	76
A_EVENT_CLOSE_FILE	10
A_EVENT_CREATE_FILE	12
A_EVENT_DELETE_FILE	14
A_EVENT_OPEN_FILE	27
A_EVENT_PURGE_FILE	214
A_EVENT_READ_FILE	42
A_EVENT_RENAME_MOVE_FILE	44
A_EVENT_SALVAGE_FILE	46
A_EVENT_WRITE_FILE	57

A_EVENT_WRITE_FILE	57
A_EVENT_MODIFY_ENTRY	25
A_EVENT_SCAN_DELETED	215
A_EVENT_SCAN_VOL_USER_REST	238
A_EVENT_SET_COMP_FILE_SIZE	242

Parent Topic:

Event Bits Tables

Message Events Table

The following table lists the bits defined in the message event bitmap.

Table auto. Message Events

Event Type ID	Event ID
A_EVENT_BROADCAST_TO_CONSOLE	207
A_EVENT_DISABLE_BROADCASTS	204
A_EVENT_ENABLE_BROADCASTS	206
A_EVENT_GET_BROADCAST_MESSAGE	205
A_EVENT_SEND_BROADCAST_MESSAGE	208

Parent Topic:

Event Bits Tables

QMS Events Table

The following table lists the bits defined in the QMS event bitmap.

Table auto. QMS Events

Event Type ID	Event ID
A_EVENT_Q_JOB_FROM_LIST	231
A_EVENT_Q_JOB_LIST	230
A_EVENT_Q_JOB_SIZE	229

A_EVENT_MOVE_Q_JOB	233
A_EVENT_Q_ATTACH_SERVER	28
A_EVENT_Q_CREATE	29
A_EVENT_Q_CREATE_JOB	30
A_EVENT_Q_DESTROY	31
A_EVENT_Q_DETACH_SERVER	32
A_EVENT_Q_EDIT_JOB	33
A_EVENT_Q_JOB_FINISH	34
A_EVENT_Q_JOB_SERVICE	35
A_EVENT_Q_JOB_SERVICE_ABORT	36
A_EVENT_Q_SWAP_RIGHTS	41
A_EVENT_Q_REMOVE_JOB	37
A_EVENT_Q_SET_JOB_PRIORITY	38
A_EVENT_Q_SET_STATUS	39
A_EVENT_Q_START_JOB	40
A_EVENT_READ_Q_JOB_ENTRY	232
A_EVENT_MOVE_Q_JOB	233
A_EVENT_READ_Q_STATUS	234
A_EVENT_READ_Q_SERVER_STATUS	235
A_EVENT_SET_Q_SERVER_STATUS	261

Parent Topic:

Event Bits Tables

Reading Auditing Event Records

To begin reading event records, call **NWADOpenRecordFile**. This function opens the record file and allocates a *recordHandle*. You can then read the event records one at a time by calling **NWADReadRecord**. Don't attempt to read the audit file directly. Since the file is compressed it must be read only with Auditing.

Resetting the audit file clears current records and stores them in an old audit file.

Parent Topic:

Auditing Event Records

Related Topics:

Fields of Auditing Event Records

Scope of Auditing Events

The event bitmap doesn’t single out specific users to be audited unless you are using NetWare® 4.11 and user restrictions is turned on using the Auditcon utility.

For previous versions of NetWare 4, once an event is set, every occurrence of the event is audited regardless of who performs it. For example, if you set the Delete Bindery Property Event bit, then an event record is generated every time a user deletes a property from an object. File events are an exception. You can configure file auditing to target specific files and users. For more information see Auditing File Events (NetWare 4.x).

If you are interested only in events involving specific items, you can filter the information you read from the audit file by searching specific fields in the event records. For example, you could search for all event records in which a particular user deleted properties from a particular object.

Parent Topic:

The Audit File Configuration Header Event Bitmap

Server Events Table

The following table lists the bits defined in the server event bitmap.

Table auto. Server Events

Event Type ID	**Event ID**
A_EVENT_CHANGE_DATE_TIME	7
A_EVENT_CONVERT_PATH_TO_ENTRY	259
A_EVENT_DISABLE_LOGIN	243
A_EVENT_DISABLE_ TTS	245
A_EVENT_DOWN_SERVER	18
A_EVENT_ENABLE_LOGIN	244
A_EVENT_ENABLE_TTS	246
A_EVENT_GET_CONN_OPEN_FILES	250
A_EVENT_GET_CONN_SEMS	256

A_EVENT_GET_CONN_TASKS	249
A_EVENT_GET_CONN_USING_FILE	251
A_EVENT_GET_LOG_REC_INFO	255
A_EVENT_GET_LOG_REC_CONN	254
A_EVENT_GET_REMAIN_OBJ_DISK_SPC	248
A_EVENT_GET_PHYS_REC_LOCKS_CONN	252
A_EVENT_GET_PHYS_REC_LOCKS_FILE	253
A_EVENT_GET_SEM_INFO	257
A_EVENT_GET_DISK_UTILIZATION	240
A_EVENT_MAP_DIR_TO_PATH	258
A_EVENT_VOLUME_DISMOUNT	56
A_EVENT_VOLUME_MOUNT	55
A_EVENT_SEND_CONSOLE_BROADCAST	247
A_EVENT_CONSOLE_COMMAND	262
A_EVENT_DESTROY_SERVICE_CONN	260
A_EVENT_VERIFY_SERIAL	239
A_EVENT_VOLUME_DISMOUNT	56
A_EVENT_VOLUME_MOUNT	55

Parent Topic:

Event Bits Tables

User Audit Property

To audit a particular user's file operations, you must assign an audit property to the user's object by calling **NWADChangeObjectProperty**. The property doesn't take a value. If the property is present, the specified user will be audited. You must specify the volume or container on which the user is being audited.

Parent Topic:

Auditing File Events (NetWare 4.x)

Related Topics:

File Events for Auditing

User Events Table

The following table lists the bits defined in the user event bitmap.

Table auto. User Events

Event Type ID	**Event ID**
A_EVENT_DISABLE_ACCOUNT	17
A_EVENT_GRANT_TRUSTEE	19
A_EVENT_LOGIN_USER	21
A_EVENT_LOGOUT_USER	23
A_EVENT_REMOVE_TRUSTEE	43
A_EVENT_USER_SPACE_RESTRIC TIONS	53
A_EVENT_USER_LOCKED	52
A_EVENT_USER_CHANGE_PASS WORD	51
A_EVENT_USER_UNLOCKED	54
A_EVENT_RENAME_USER	45

Parent Topic:

Event Bits Tables

Auditing: Functions

NWADAppendExternalRecords

Adds external audit events to an event record

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.11

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

Service: Auditing

Syntax

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

N_EXTERNAL_LIBRARY NWRCODE NWADAppendExternalRecords (
   NWCONN_HANDLE      conn,
   nuint32            auditFileObjectID,
   nuint32            vendorID,
   nuint32            numberRecords,
   pNWAuditRecord    recordsPtr);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADAppendExternalRecords
  (connectionNumber : NWCONN_HANDLE;
   auditFileObjectID : nuint32;
   vendorID : nuint32;
   numberRecords : nuint32;
   Var recordsPtr : NWAuditRecord
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

auditFileObjectID

(IN) Specifies the NetWare version audit object ID.

vendorID

(IN) Specifies the vendor workstation number assigned by Novell.

numberRecords

(IN) Specifies the number of audit event records in the NWAuditRecord structure.

recordsPtr

(IN) Points to the NWAuditRecord structure containing the records to be stored.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADAppendExternalRecords will only work with NetWare 4.11. For a list of event bits, see Event Bits Tables.

NCP Calls

104 52 External Audit Append To File

NWADChangeObjectProperty

Adds or removes the audit property for a user on a specified volume or container

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADChangeObjectProperty (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   nuint32         objectID,
   nuint8          auditFlag);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADChangeObjectProperty
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   objectID : nuint32;
   auditFlag : nuint8
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare® server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

objectID

(IN) Specifies the object ID number on which to change the audit property. If the object to be changed is a volume, *objectID* specifies the user ID; it specifies the Directory Services object number if a Directory Services object is to be changed.

auditFlag

(IN) Specifies whether the object will be audited: 1 = the object will be audited; 0 = the object will not be audited.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADChangeObjectProperty requires a second-level password when enabled.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 02	Add User Audit Property
0x2222 88 06	Delete Audit Property

0x2222 104 221   Audit Change Object Audited

See Also

NWADInitLevelTwoPassword, **NWADIsObjectAudited**, **NWADLogin**, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADChangePassword

Changes the auditor's password for a specified volume or container
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADChangePassword (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   pnuint8         newPassword,
   nuint8          level);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADChangePassword
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   Var newPassword : nuint8;
   level : nuint8
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

newPassword

(IN) Points to a NULL-terminated character string containing the new password.

level

(IN) Specifies which password to change; for example, 2 = level two password.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x89D7	PASSWORD_NOT_UNIQUE
0x89D8	PASSWORD_TOO_SHORT
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

If **NWADChangePassword** fails, the original password is still valid.

NWADChangePassword is only supported for NetWare 4.1 and above. To call **NWADChangePassword** under NetWare 4.11, a password has to be set and the user who set the password cannot be a password user.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 04 Change Auditor Volume Password

0x2222 88 18	Change Audit Level Two Volume Password
0x2222 88 19	Get Auditing Flags
0x2222 104 203	Directory Services Change Audit Password
0x2222 104 215	Directory Services Change Audit Level Two Password
0x2222 104 216	Get Auditing Flags

See Also

NWADLogin, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADCheckAccess

Checks to see if the auditor has auditor access

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADCheckAccess (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADCheckAccess
   (conn : NWCONN_HANDLE;
    auditIDType : nuint32;
    auditID : nuint32
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x0001	No Audit Access
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

If zero (0) is returned, the user has auditor access and is currently logged in through **NWADLogin**.

If one (1) is returned, the user does not have auditor access.

In NetWare 4.11, you can only call **NWADLogin** once, which will set the auditor access on the file server. If subsequent calls are made, an error will be returned.

The second level password is only supported under NetWare 4.1. NetWare 4.11 does not use second level passwords.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 05	Change Auditor Access
0x2222 104 204	Directory Services Check Auditor Access

See Also

NWADCheckLevelTwoAccess, **NWADLogin**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADCheckLevelTwoAccess

Checks to see if the auditor has level two access
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADCheckLevelTwoAccess (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADCheckLevelTwoAccess
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x0001	No Audit Access
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADCheckLevelTwoAccess requires a level two password to be initialized.

If zero (0) is returned, the user has level two access.

If one (1) is returned, the user does not have audit level two access.

NWADCheckLevelTwoAccess is only supported on a NetWare 4.1 file server. NetWare 4.11 does not use second level passwords.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 22	Check Level Two Access
0x2222 104 219	Directory Services Check Audit Level Two Access

See Also

NWADCheckAccess, **NWADInitLevelTwoPassword**, **NWADLogin**, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADClose

Frees the *auditHandle* allocated by **NWADOpen** and NULL the pointer to the audit handle

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADClose (
   pnptr   auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADClose
  (Var auditHandle : nptr
) : NWRCODE;
```

Parameters

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL

Remarks

All *auditHandle* parameters allocated by **NWADOpen** must be freed by calling **NWADClose**.

See Also

NWADOpen

NWADCloseOldFile

Closes the old auditing file automatically opened by the system when the volume or container is mounted or auditing is enabled

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADCloseOldFile (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADCloseOldFile
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADCloseOldFile requires a second-level password when enabled.

The old auditing file is kept open by the operating system. If a file needs to be copied or deleted, close it by calling **NWADCloseOldFile** first.

NWADCloseOldFile is only supported on NetWare 4.1. **NWADCloseOldFile** cannot be called from NetWare 4.11.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 20 Close Old Audit File

0x2222 104 217 Directory Services Close Old Audit File

See Also

NWADDeleteFile, **NWADInitLevelTwoPassword**, **NWADLogin**, **NWADOpen**, **NWADResetFile**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADCloseRecordFile

Frees the *recordHandle* allocated by **NWADOpenRecordFile**
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADCloseRecordFile (
   pnptr   recordHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADCloseRecordFile
  (Var recordHandle : nptr
) : NWRCODE;
```

Parameters

recordHandle
(IN/OUT) Points to the record handle to free.

Return Values

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

0x0000	SUCCESSFUL

Remarks

All resources used in the *recordHandle*s allocated by **NWADOpenRecordFile** are freed by calling **NWADCloseRecordFile**.

See Also

NWADOpenRecordFile, **NWADReadRecord**

NWADDeleteFile

Deletes the old auditing file on the specified volume or container
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADDeleteFile (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADDeleteFile
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADDeleteFile requires a second level password when enabled.

NWADDeleteFile is only supported on NetWare 4.1. Call **NWADDeleteOldFile** for NetWare 4.1 and above for future compatibility.

NWADDeleteFile cannot be called from NetWare 4.11.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 21 Delete Old Audit File

See Also

NWADCloseOldFile, **NWADInitLevelTwoPassword**, **NWADLogin**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADDeleteOldFile

Deletes an old file in the file list
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADDeleteOldFile
  (NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   nuint32         fileCode);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADDeleteOldFile
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   fileCode : nuint32
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

fileCode

(IN) Specifies the number of the file on the file list to delete (0-15).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

If you want to delete the current file, call **NWADResetFile** to move the current file into the file list. Then call **NWADDeleteOldFile** to delete that file.

The file number zero (0) indicates the current history file; file numbers 1-15 indicate old files that can be deleted by calling **NWADDeleteOldFile**.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 26 Delete Old Audit File

0x2222 104 225 Delete Old Audit File

See Also

NWADOpen, **NWADResetFile**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADDisable

Disables auditing on a specified volume or container
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADDisable (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADDisable
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADDisable requires a second level password when enabled.

You will lose audit access on the NetWare server after calling **NWADDisable** because auditing is disabled. To do more auditing, you must log in again by calling **NWADLogin**.

If a password user disables auditing, that same user cannot enable auditing again in NetWare 4.11. The only way auditing may be enabled again is through the authorized auditing Directory Services objects.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 07   Disable Volume Auditing

0x2222 104 206   Directory Services Disable Volume Auditing

See Also

NWADEnable, **NWADInitLevelTwoPassword**, **NWADLogin**, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADEnable

Enables auditing on the specified Directory Services container or volume

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADEnable (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADEnable
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled

Remarks

If auditing has never been initialized on the volume, call **NWADLogin** first; 0x8997 will be returned. Then, call **NWADEnable**.

If the user is not SUPERVISOR equivalent, **NWADEnable** will fail the first time it is called.

After **NWADEnable** has been called successfully, the user must log in again by calling **NWADLogin** to have access to auditing.

For NetWare 4.11, a different approach must be followed to enable auditing. You must log into NetWare through Directory Services and have the necessary rights to create objects and add attributes. You may then add auditor access by adding the AFO attributes to a user object and assigning it to a volume or container. This user can then enable auditing on the volume or container.

A password user on NetWare 4.11 cannot enable auditing.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 104 207	Directory Services Enable Container Volume Auditing

See Also

NWADDisable, **NWADInitLevelTwoPassword**, **NWADLogin**, **NWADOpen**

NWADOpen, NWDSAuditGetObjectID, NWGetVolumeNumber

NWADGetFileList

Returns the file list
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADGetFileList (
   NWCONN_HANDLE       conn,
   nuint32             auditIDType,
   nuint32             auditID,
   nptr                auditHandle,
   pNWAuditFileList   fileList);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADGetFileList
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   fileList : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

fileList

(OUT) Points to the structure NWAuditFileList containing the size, date, and time for the past 16 elements.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADGetFileList returns a list of the dates, times, and sizes of old audit files. The list contains a total of up to 16 (0-15) files that can be added to the list by calling **NWADResetFile**.

If the *fileCreateDateTime* field of NWAuditFileList is not zero, the *fileSize* field is valid.

The file number zero (0) indicates the current history file; file numbers 1-15 indicate old history files. The number of old files to keep is indicated by *bufferSize* in **NWADReadConfigHeader** and **NWADWriteConfigHeader** and has a maximum of 15 files.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 23 Return Audit File List

0x2222 104 222 Return Audit File List

See Also

NWADCloseRecordFile, **NWADOpen**, **NWADOpenRecordFile**, **NWADReadRecord**, **NWADResetFile**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADGetFlags

Returns the auditing flag byte

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADGetFlags (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   pnuint8         flags);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADGetFlags
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   Var flags : nuint8
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

flags

(OUT) Points to a byte where the flags can be returned.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8998	VOLUME_DOES_NOT_EXIST

Remarks

NWADGetFlags returns the auditing flag byte, whose contents follow:

```
0x01 = DiscardAuditRcdsOnErrorFlag
0x02 = ConcurrentVolAuditorAccess
0x04 = DualLevelPasswordsActive
0x08 = BroadcastWarningsToAllUsers
0x10 = LevelTwoPasswordSet
0x20 = ArchiveAuditFileOnErrorFlag
```

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 19	Return Audit Flags
0x2222 104 216	Directory Services Return Audit Flags

See Also

NWADLogin, **NWADOpen**, **NWADReadConfigHeader**, **NWADWriteConfigHeader**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADGetStatus

Returns the audit information and status of the specified volume or container

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADGetStatus (
   NWCONN_HANDLE     conn,
   nuint32           auditIDType,
   nuint32           auditID,
   pNWAuditStatus    auditStatus,
   nuint16           bufferSize);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADGetStatus
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   Var auditStatus : NWAuditStatus;
   bufferSize : nuint16
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditStatus

(OUT) Points to NWAuditStatus containing fields for the information to be returned.

bufferSize

(IN) Specifies the size of the memory space.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8998	VOLUME_DOES_NOT_EXIST
0x89F2	Audit Password Enabled

Remarks

The *historyRecordCount* of the NWAuditStatus structure will remain zero (0) because the history records are kept inside the audit file.

For NetWare 4.11, if **NWADGetStatus** returns 0x89F2, the user is not allowed auditor access. However, the NWAuditStatus structure will still be filled. You should check the *auditingFlags* field for a value of one (1) which indicates passwords are allowed. If the value is one (1), **NWADLogin** can then be called with a valid password. Call **NWADCheckAccess** to set the audit access bit on the server side. Subsequent calls will then be enabled for password users on NetWare 4.11.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 01	Return Volume Audit Status
0x2222 104 200	Directory Services Return Volume Audit Status

See Also

NWDSAuditGetObjectID, **NWGetVolumeNumber**

NWADInitLevelTwoPassword

Enables auditor level two access on a specified volume
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADInitLevelTwoPassword (
   nptr       auditHandle,
   pnuint8    password);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADInitLevelTwoPassword
   (auditHandle :nptr;
   Var password : nuint8
) : NWRCODE;
```

Parameters

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

password

(IN) Points to a NULL-terminated character string containing the password.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8984	Auditing Not Supported
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

auditHandle is initialized and set up for level two access.

NWADInitLevelTwoPassword does not modify auditing flags and does not verify the password.

Call **NWADInitLevelTwoPassword** to set up the Directory Service Level Two Password also.

NWADCheckLevelTwoAccess can be called to verify if the password is valid on NetWare 4.1. (**NWADCheckLevelTwoAccess** is not supported on NetWare 4.11.)

See Also

NWADLogin, **NWADCheckLevelTwoAccess**

NWADIsObjectAudited

Checks to see if specified object or user is being audited
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADIsObjectAudited (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nuint32         userObjectID);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADIsObjectAudited
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   userObjectID : nuint32
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

userObjectID

(IN) Specifies the object ID to be checked if it is being audited.

Return Values

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

0x0000	SUCCESSFUL---User is not being audited.
0x0001	SUCCESSFUL---User is being audited.

Remarks

NWADIsObjectAudited returns 0x0000 if the user is not being audited and 0x0001 if the user is being audited.

userObjectID must be byte-swapped to the same format in which the server stores it.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 09	Is User Audited
0x2222 104 220	Is Object Audited

See Also

NWADChangeObjectProperty, **NWADLogin**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADLogin

Enables auditor access on a specified container or volume

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADLogin (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   pnuint8         password);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADLogin
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   Var password : nuint8
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

password

(IN) Points to the address of a NULL-terminated character string containing the password.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

Calling **NWADLogin** is the first step to any auditing function.

auditHandle is initialized and setup for future auditing API calls; *auditHandle* must be allocated by the program.

Only a level one password is authenticated with **NWADLogin**.

If auditing has never been initialized on the Container, call **NWADLogin** first; AUDITING_NOT_ENABLED will be returned. Then, call **NWADEnable**.

If the user is not SUPERVISOR equivalent, **NWADEnable** will fail the first time it is called.

After calling **NWADEnable** successfully, the user must log in again by calling **NWADLogin** to have access to auditing.

Once auditing has been enabled, the user does not have to be SUPERVISOR equivalent, but must know the auditor password.

NetWare 4.11 does not use a password unless a password has been set by calling **NWADSetPassword**. Call **NWADGetStatus** to determine if a password has been set.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 03	Add Auditor Access
0x2222 88 19	Get Auditing Flags
0x2222 104 202	Directory Services Add Auditor Access
0x2222 104 216	Get Auditing Flags

See Also

NWADCheckAccess, **NWADEnable**, **NWADGetStatus**, **NWADInitLevelTwoPassword**, **NWADLogout**, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADLogout

Removes auditor access from a volume or container while*auditHandle* resets to NULL

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADLogout (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADLogout
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADLogout must be called every time an application logs in as an auditor.

NWADLogout must be called every time an application logs in as an auditor on the Directory Services container.

NWADLogout cannot be used on NetWare 4.11 unless a password has been set by calling **NWADSetPassword**. Call **NWADGetStatus** to determine if a password has been set.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 13	Remove Auditor Access
0x2222 104 211	Directory Services Remove Auditor Access

See Also

NWADCheckAccess, **NWADGetStatus**, **NWADLogin**, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADOpen

Allocates *auditHandle* for use in other Auditing functions
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADOpen (
   NWCONN_HANDLE      conn,
   nuint32            auditIDType,
   nuint32            auditID,
   pnptr              auditHandle,
   pNWADOpenStatus    openStatus);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADOpen
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   Var auditHandle : nptr;
   Var openStatus : NWADOpenStatus
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the auditID parameter.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the volume number for volume auditing or the object ID of the DS Audit File Object for container auditing.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

openStatus

(OUT) Points to the NWADOpenStatus structure containing the file status.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWGetVolumeNumber can be called to query the volume number.

NWDSAuditGetObjectID can be called to query the object ID.

See Also

NWADClose, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADOpenRecordFile

Opens the record file

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADOpenRecordFile (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   nint16          fileCode,
   pnptr           recordHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADOpenRecordFile
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   fileCode : nint16;
   Var recordHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

fileCode

(IN) Specifies the file number to open. -1 specifies the current file from which to read records while 0-15 specifies old files from which to read records.

recordHandle

(OUT) Points to an allocated record handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 24 Init Audit File Read

0x2222 104 223 Init Audit File Read

See Also

NWADCloseRecordFile, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADReadBitMap

Reads the audit bitmap to see what is being audited

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADReadBitMap (
   NWCONN_HANDLE          conn,
   nuint32                volumeID,
   NWAuditBitMap N_FAR  *buffer,
   nuint16                bufferSize);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADReadBitMap
  (conn : NWCONN_HANDLE;
   volumeID : nuint32;
   Var buffer : NWAuditBitMap;
   bufferSize : nuint16
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volumeID

(IN) Specifies the volume number of the audited volume.

buffer

(OUT) Points to NWAuditBitMap defining what is being audited.

bufferSize

(IN) Specifies the size of NWAuditBitMap expected. The entire bitmap needs to be received at once.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWAuditMap is a 512-bit stream. If a bit is set to one (1), the corresponding item found in auditBitMapIDs enumeration is being audited.

NWAuditBitMap may also be read by calling **NWReadAuditConfigHeader**.

When a bit in auditBitMapID is set to one, it generates an event and saves it in the audit file. The definition of the set bits follow:

Bit	Function	Description
A_BIT_GOPEN_FILE	Global Open	Specifies all file opens are audited
A_BIT_IOPEN_FILE	Intersection Open	Specifies the intersection of events by a user or a file open is audited
A_BIT_UOPEN_FILE	Union Open	Specifies the union of events by a user and a file open is audited.

auditBitMapID lists the bits defined by the event bitmap in the audit file configuration header. The first 32 bits are reserved for Directory Services for container auditing. For a list of event bits, see Event Bits Tables.

NWADReadBitMap is only supported on NetWare 4.1. It is not supported on NetWare 4.11.

NCP Calls

0x2222 88 10 Read Audit Bit Map

See Also

NWADReadConfigHeader, **NWADWriteBitMap**, **NWADLogin**

NWADReadConfigHeader

Returns the configuration header from the auditing file on a specified volume or container

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADReadConfigHeader (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle,
   nptr            buffer,
   nuint16         bufferSize);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADReadConfigHeader
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   buffer : nptr;
   bufferSize : nuint16
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

buffer

(OUT) Points to an array in which data is saved.

bufferSize

(IN) Specifies the size of the configuration header for saving data.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

volumeAuditEventBitMap in NWConfigHeader may also be read by calling **NWReadAuditingBitMap**.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

For a list of event bits, see Event Bits Tables.

NCP Calls

0x2222 88 11 Read Audit Config Hdr

0x2222 104 209 Directory Services Read Audit Configuration Header

See Also

NWADLogin, NWADReadBitMap, NWADWriteConfigHeader, NWDSAuditGetObjectID, NWGetVolumeNumber

NWADReadRecord

Reads a specified record
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADReadRecord (
   nptr        recordHandle,
   nuint16     maxSize,
   nint16      direction,
   pnuint8     buffer,
   pnuint16    bufferSize,
   pnuint8     eofFlag,
   pnuint32    offsetPtr);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADReadRecord
  (recordHandle : nptr;
   maxSize : nuint16;
   direction : nint16;
   Var buffer : nuint8;
   Var bufferSize : nuint16;
   Var eofFlag : nuint8;
   Var offsetPtr : nuint32
) : NWRCODE;
```

Parameters

recordHandle

(IN) Specifies the record handle allocated in **NWADOpenRecordFile**.

maxSize

(IN) Specifies the size of the buffer passed into the call. **NWADReadRecord** will write beyond the end of the specified buffer size. Typically, size is 512 bytes.

direction

(IN) Specifies whether to get the previous or the next record:

-1 = Get previous record
1 = Get next record

buffer

(IN/OUT) Points to a buffer to contain the record.

bufferSize

(OUT) Points to the size of data contained in the buffer.

eofFlag

(OUT) Points to a flag indicating whether the end of the file has been reached:

1 = End of file
0 = More file to be read

offsetPtr

(IN) Points to a book marker indicating where the previously read record is located.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

Either an end of file flag or -1 is a valid signal to **NWADReadRecord** to stop reading from the specified file.

NCP Calls

0x2222 88 25 Read Auditing File
0x2222 104 224 Read Auditing File

See Also

NWADOpen, **NWADOpenRecordFile**, **NWADCloseRecordFile**

NWADResetFile

Resets the audit file on a specified container or volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADResetFile (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID,
   nptr            auditHandle);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADResetFile
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

The original file is saved to the OLD file and a new audit file is set up to store audit events.

NWADResetFile requires a level two password when enabled.

The name of the data file is containerID.DAF. When the file is reset, it is renamed to containID.OAF. These files are hidden, and reside in a hidden directory called _NetWare on a NetWare server having a writable replica of the container probably where the object resides.

NWADResetFile is only supported on NetWare 4.1; NetWare 4.11 does not support **NWADResetFile**.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 14	Reset Audit File
0x2222 104 212	Directory Services Reset Audit File

See Also

NWADInitLevelTwoPassword, **NWADLogin**, **NWADOpen**, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADRestartVolumeAuditing

Restarts auditing for volumes only
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADRestartVolumeAuditing (
   NWCONN_HANDLE   conn,
   nuint32         auditIDType,
   nuint32         auditID);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADRestartVolumeAuditing
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL

Remarks

NWADRestartVolumeAuditing is for NetWare 4.11 only and is not supported on NetWare 4.1.

NWADRestartVolumeAuditing should be called when the history file has reached the size limit or a volume is full. The auditor (not a password auditor) should access the volume and correct the situation, and then call **NWADRestartVolumeAuditing** to restart auditing on the volume. Other users may then access the volume.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 30   Restart Volume Auditing

See Also

NWDSAuditGetObjectID, **NWGetVolumeNumber**

NWADSetPassword

Sets a password only on NetWare 4.11
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADSetPassword
   (NWCONN_HANDLE  conn,
    nuint32        auditIDType,
    nuint32        auditID,
    nptr           auditHandle,
    pnuint8        newPassword);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADSetPassword
   (conn : NWCONN_HANDLE;
    auditIDType : nuint32;
    auditID : nuint32;
    auditHandle : nptr;
    Var newPassword : nuint8
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

newPassword

(IN) Points to the new password to be set.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x89D7	PASSWORD_NOT_UNIQUE
0x89D8	PASSWORD_TOO_SHORT
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADSetPassword is only supported on the next released version after NetWare 4.1.

NWADSetPassword allows auditing to be accessed by a password user who does not have Audit File Object access rights.

Password users have limited access in error conditions and setting up auditing.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 19	Get Audit Flags
0x2222 88 31	Set Audit Password
0x2222 104 216	Get Audit Flags
0x2222 104 229	Set Audit Password

See Also

NWADChangePassword, **NWDSAuditGetObjectID**, **NWGetVolumeNumber**

NWADWriteBitMap

Writes the audit bitmap definition of what is being audited
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.1 and above
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Auditing

Syntax

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

N_EXTERN_LIBRARY (NWRCODE) NWADWriteBitMap (
   NWCONN_HANDLE         conn,
   nuint32               volumeID,
   nptr                  auditHandle,
   NWAuditBitMap N_FAR  *buffer);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADWriteBitMap
  (conn : NWCONN_HANDLE;
   volumeID : nuint32;
   auditHandle : nptr;
   Var buffer : NWAuditBitMap
) : NWRCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

volumeID
(IN) Specifies the volume number of the audited volume.

auditHandle
(IN) Points to the *auditHandle* allocated by **NWADOpen**.

buffer
(IN) Points to NWAuditBitMap defining what is being audited.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADWriteBitMap requires a second level password when enabled.

This constant defines the number of bits in the event bitmap for volumes:

```
NW_AUDIT_NUMBER_EVENT_BITS = 512
```

The buffer is a 512-bit stream. If a bit is set to one (1), the corresponding item found in auditBitMapIDs (nwaudit.h) is being audited.For a list of event bits, see Event Bits Tables.

If auditing is not enabled, NO_DISK_LEFT_FOR_SPOOL_FILE is returned; if auditing is not supported, NO_CREATE_PRIVILEGES is returned.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 16 Write Auditing Bit Map

See Also

NWADLogin, **NWADOpen**, **NWADWriteConfigHeader**, **NWADReadBitMap**, **NWADReadConfigHeader**, **NWADInitLevelTwoPassword**

NWADWriteConfigHeader

Saves the configuration header for the auditing file on a specified volume or container

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

N_EXTERN_LIBRARY NWRCODE NWADWriteConfigHeader (
   NWCONN_HANDLE     conn,
   nuint32           auditIDType,
   nuint32           auditID,
   nptr              auditHandle,
   pNWConfigHeader   buffer);
```

Pascal Syntax

```
#include <nwaudit.inc>

Function NWADWriteConfigHeader
  (conn : NWCONN_HANDLE;
   auditIDType : nuint32;
   auditID : nuint32;
   auditHandle : nptr;
   Var buffer : NWConfigHeader
) : NWRCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

auditIDType

(IN) Specifies the type of the object to be audited.

0 AUDIT_ID_IS_VOLUME indicates volume auditing
1 AUDIT_ID_IS_CONTAINER indicates container auditing

auditID

(IN) Specifies the identification of the object to be audited.

auditHandle

(IN) Points to the *auditHandle* allocated by **NWADOpen**.

buffer

(IN) Points to NWConfigHeader containing data.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8983	Auditing Hardware Error
0x8984	Auditing Not Supported
0x8997	Auditing Not Enabled
0x8998	VOLUME_DOES_NOT_EXIST
0x89DE	PASSWORD_HAS_EXPIRED_NO_GRACE

Remarks

NWADWriteConfigHeader requires a second level password when enabled.

If auditing is not enabled, NO_DISK_LEFT_FOR_SPOOL_FILE is returned; if auditing is not supported, NO_CREATE_PRIVILEGES is returned.

If *auditIDType* is set to AUDIT_ID_IS_VOLUME to indicate volume auditing, **NWGetVolumeNumber** can be called to get the volume number of the audit file object.

If *auditIDType* is set to AUDIT_ID_IS_CONTAINER to indicate container auditing, **NWDSAuditGetObjectID** can be called to get the Directory Service object ID of the audit file object.

NCP Calls

0x2222 88 17	Write Audit Config Hdr
0x2222 88 19	Get Auditing Flags
0x2222 104 214	Directory Services Write Audit CNT Config Hdr
0x2222 104 216	Get Auditing Flags

See Also

NWADInitLevelTwoPassword, NWADLogin, NWADReadBitMap, NWADReadConfigHeader, NWADWriteBitMap, NWDSAuditGetObjectID, NWGetVolumeNumber

NWGetNWADVersion

Returns the version information about the NWAD library

NetWare Server: 4.1 and above

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

Service: Auditing

Syntax

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

void N_API NWGetNWADVersion (
   pnuint8   majorVersion,
   pnuint8   minorVersion,
   pnuint8   revisionLevel,
   pnuint8   betaReleaseLevel);
```

Pascal Syntax

```
#include <nwaudit.inc>

Procedure NWGetNWADVersion
  (Var majorVersion : nuint8;
   Var minorVersion : nuint8;
   Var revisionLevel : nuint8;
   Var betaReleaseLevel : nuint8
);
```

Parameters

majorVersion

(OUT) Points to the major version of the library. For example, 4 would be returned for NetWare 4.1.

minorVersion

(OUT) Points to the minor version of the library. For example, .1 would be returned for NetWare 4.1.

revisionLevel

(OUT) Points to the revision version of the library. For example, c would be returned for NetWare 3.11c.

betalReleaseLevel

(OUT) Points to the beta release version of the library.

Return Values

Management Service Group

None

Auditing: Structures

NWADOpenStatus

Service: Auditing
Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint32   auditingStatus;
   nuint32   isTrustedNetWare;
   nuint32   trustedNetWareStatus;
   nuint32   reserved1;
   nuint32   reserved2;
   nuint32   reserved3;
   nuint32   reserved4;
} NWADOpenStatus;
```

Pascal Structure

```
Defined in nwaudit.inc

 NWADOpenStatus  = Record
    auditingStatus : nuint32;
    isTrustedNetWare : nuint32;
    trustedNetWareStatus : nuint32;
    reserved1 : nuint32;
    reserved2 : nuint32;
    reserved3 : nuint32;
    reserved4 : nuint32
  End;
```

Fields

auditingStatus

isTrustedNetWare

Specifies if trusted NetWare is running:

0 False
1 True

trustedNetWareStatus

Specifies if a password if needed:

1 Requires user password

reserved1

Reserved by Novell for future use.

reserved2

Reserved by Novell for future use.

reserved3

Reserved by Novell for future use.

reserved4

Reserved by Novell for future use.

NWAuditBitMap

Defines an array of bytes large enough to contain the number of bits in a configuration header's event bitmap

Service: Auditing

Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint8   bitMap[NW_AUDIT_NUMBER_EVENT_BITS/8];
} NWAuditBitMap;
```

Pascal Structure

```
Defined in nwaudit.inc

 NWAuditBitMap = Record
      bitMap : Array[0..NW_AUDIT_NUMBER_EVENT_BITS DIV 8] Of nuint8
   End;
```

Fields

bitMap

Specifies if the event is audited as defined in the nwadevnt.h file:

0 Event is not audited

1 Event is audited

NWAuditBitMapTNW

Defines an array of bytes large enough to contain the number of bits in a configuration header's event bitmap

Service: Auditing

Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint8   bitMap[NW_AUDIT_NUMBER_EVENT_BITS_TNW/8];
} NWAuditBitMapTNW;
```

Pascal Structure

```
Defined in nwaudit.inc

NWAuditBitMapTNW = Record
      bitMap : Array[0..NW_AUDIT_NUMBER_EVENT_BITS_TNW DIV 8] Of nuint8
   End;
```

Fields

bitMap

NWAuditFileList

Service: Auditing
Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint32   fileCreateDateTime[16];
   nuint32   fileSize[16];
} NWAuditFileList;
```

Pascal Structure

```
Defined in nwaudit.inc

 NWAuditFileList = Record
      fileCreateDateTime : nuint32;
      fileSize : nuint32
   End;
```

Fields

fileCreateDateTime

Specifies the time the file was saved by calling the **NWADResetAuditFile** function.

fileSize

Specifies the size of the file.

NWAuditRecord

Service: Auditing
Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint32   recordLength;
   pnuint8   record;
} NWAuditRecord;
```

Pascal Structure

```
Defined in nwaudit.inc

 NWAuditRecord = Record
    recordLength : nuint32;
    aRecord : pnuint8
  End;
```

Fields

recordLength

Specifies the size of the data to send.

record

Points to the data to send.

NWAuditStatus

Service: Auditing
Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint16   auditingVersionDate;
   nuint16   auditFileVersionDate;
   nuint32   auditingEnabledFlag;
   nuint32   auditFileSize;
   nuint32   modifiedCounter;
   nuint32   auditFileMaxSize;
   nuint32   auditFileSizeThreshold;
   nuint32   auditRecordCount;
   nuint32   auditingFlags;
} NWAuditStatus;
```

Pascal Structure

```
Defined in nwaudit.inc

NWAuditStatus = Record
    auditingVersionDate : nuint16;
    auditFileVersionDate : nuint16;
    auditingEnabledFlag : nuint32;
    auditFileSize : nuint32;
    modifiedCounter : nuint32;
    auditFileMaxSize : nuint32;
    auditFileSizeThreshold : nuint32;
    auditRecordCount : nuint32;
    auditingFlags : nuint32
 End;
```

Fields

auditingVersionDate

Specifies the file server version data of auditing.

auditFileVersionDate

auditingEnabledFlag

Specifies whether auditing is enabled:

0 Disabled
1 Enabled

auditFileSize

auditFileSize

Specifies the size of the current audit file.

modifiedCounter

Specifies the number of times auditing has been modified.

auditFileMaxSize

Specifies the maximum size for the audit file.

auditFileSizeThreshold

Specifies the size of the file when auditing starts to send warning messages.

auditRecordCount

Specifies the total number of events in the audit file.

auditingFlags

NWConfigHeader

Stores information associated with a volume's audit file configuration header

Service: Auditing

Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint16         fileVersionDate;
   nuint8          auditFlags;
   nuint8          errMsgDelayMinutes;
   nuint8          reserved1[16];
   nuint32         auditFileMaxSize;
   nuint32         auditFileSizeThreshold;
   nuint32         auditRecordCount;
   nuint32         historyRecordCount;
   nuint8          reserved2[16];
   nuint32         reserved3[3];
   nuint8          auditEventbitMap[NW_AUDIT_NUMBER_EVENT_BITS/8];
   nuint32         auditFileCreationDateTime;
   nuint8          reserved4[8];
   nuint16         auditFlags2;
   nuint16         fileVersionDate2;
   nuint8          fileArchiveDays;
   nuint8          fileArchiveHour;
   nuint8          numOldAuditFilesToKeep;
   nuint8          reserved5;
   nuint32         headerChecksum;
   nuint32         headerModifiedCounter;
   nuint32         reserved6;
   nuint8          newbitMap[64];
   nuint8          reserved7[64];
} NWConfigHeader;
```

Pascal Structure

```
Defined in nwaudit.inc

NWConfigHeader = Record
      fileVersionDate : nuint16;
      auditFlags : nuint8;
      errMsgDelayMinutes : nuint8;
      reserved1 : Array[0..15] Of nuint8;
      auditFileMaxSize : nuint32;
      auditFileSizeThreshold : nuint32;
      auditRecordCount : nuint32;
      historyRecordCount : nuint32;
      reserved2 : Array[0..15] Of nuint8;
      reserved3 : Array[0..2] Of nuint32;
      auditEventBitMap : Array[0..NW_AUDIT_NUMBER_EVENT_BITS DIV 8] Of
      auditFileCreationDateTime : nuint32;
      reserved4 : Array[0..7] Of nuint8;
      auditFlags2 : nuint16;
      fileVersionDate2 : nuint16;
      fileArchiveDays : nuint8;
      fileArchiveHour : nuint8;
      numOldAuditFilesToKeep : nuint8;
      reserved5 : nuint8;
      headerChecksum : nuint32;
      headerModifiedCounter : nuint32;
      (* Trusted NetWare uses the following two fields *)
      reserved6 : nuint32;
      (*Tusted NetWare uses this bitmap instead of
         volumeAuditEventBitMap above       *)
      newBitMap : Array[0..63] Of nuint8;
      reserved7 : Array[0..63] Of nuint8
   End;
```

Fields

fileVersionDate

Indicates the current version of the audit file.

auditFlags

Indicates the set of bit flags controlling the audit:

errMsgDelayMinutes

Indicates the number of minutes to delay between error messages.

reserved1

Indicates the maximum allowable size of the audit file.

auditFileMaxSize

Indicates the maximum allowable size of the audit file.

auditFileSizeThreshold

Indicates the size at which users are notified that the audit file is approaching its maximum size.

auditRecordCount

Indicates the number of records in the audit file.

historyRecordCount

Indicates the number of records in the audit history file (found only in the volume configuration).

reserved2

reserved3

auditEventBitMap

auditFileCreationDateTime

reserved4

auditFlags2

fileVersionDate2

fileArchiveDays

fileArchiveHour

numOldAuditFilesToKeep

reserved5

headerChecksum

headerModifiedCounter

reserved6

newBitMap
Trusted NetWare uses this bitmap instead of *auditEventBitMap*.

reserved7
Trusted NetWare uses this field.

NWDSContainerConfigHdr

Stores information associated with a container's audit file configuration header

Service: Auditing

Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint16     fileVersionDate;
   nuint8      auditFlags;
   nuint8      errMsgDelayMinutes;
   nuint32     containerID;
   nuint32     reserved1;
   TIMESTAMP   creationTS;
   nuint32     bitMap;
   nuint32     auditFileMaxSize;
   nuint32     auditFileSizeThreshold;
   nuint32     auditRecordCount;
   nuint16     replicaNumber;
   nuint8      enabledFlag;
   nuint8      fileArchiveDays;
   nuint8      fileArchiveHour;
   nuint8      numOldFilesToKeep;
   nuint16     numberReplicaEntries;
   nuint32     auditFileCreationDateTime;
   nuint8      reserved2[8];
   nuint32     partitionID;
   nuint32     headerChecksum;
   nuint32     reserved3[4];
   nuint32     auditDisabledCounter;
   nuint32     auditEnabledCounter;
   nuint8      reserved[32];
   nuint32     hdrModifiedCounter;
   nuint32     fileResetCounter;
   nuint8      newBitMap[64];
   nuint8      reserved7[64];
} NWDSContainerConfigHdr;
```

Pascal Structure

```
Defined in nwaudit.inc

 NWDSContainerConfigHdr = Record
      fileVersionDate : nuint16;
      auditFlags : nuint8;
      errMsgDelayMinutes : nuint8;
      containerID : nuint32;
      reserved1 : nuint32;
      creationTS : TIMESTAMP;
      bitMap : nuint32;
      auditFileMaxSize : nuint32;
      auditFileSizeThreshold : nuint32;
      auditRecordCount : nuint32;
      replicaNumber : nuint16;
      enabledFlag : nuint8;
      fileArchiveDays : nuint8;
      fileArchiveHour : nuint8;
      numOldFilesToKeep : nuint8;
      numberReplicaEntries : nuint16;
      auditFileCreationDateTime : nuint32;
      reserved2 : Array[0..7] Of nuint8;
      partitionID : nuint32;
      headerChecksum : nuint32;
      reserved3 : Array[0..3] Of nuint32;
      auditDisabledCounter : nuint32;
      auditEnabledCounter : nuint32;
      reserved4 : Array[0..31] Of nuint8;
      hdrModifiedCounter : nuint32;
      (* Trusted NetWare uses the following two fields *)
      fileResetCounter : nuint32;
      (* Tusted NetWare uses this bitmap *)
      newBitMap : Array[0..63] Of nuint8;
      reserved5 : Array[0..63] Of nuint8
   End;
```

Fields

fileVersionDate

auditFlags

errMsgDelayMinutes

containerID

reserved1

creationTS

bitMap

```
enum auditBitMapIDs
{
   ADS_BIT_ADD_ENTRY               = 1,
   ADS_BIT_REMOVE_ENTRY            = 2,
   ADS_BIT_RENAME_OBJECT           = 3,
   ADS_BIT_MOVE_ENTRY              = 4,
   ADS_BIT_ADD_SECURITY_EQUIV      = 5,
   ADS_BIT_REMOVE_SECURITY_EQUIV   = 6,
   ADS_BIT_ADD_ACL                  = 7,
   ADS_BIT_REMOVE_ACL               = 8,
   A_BIT_BIND_CHG_OBJ_SECURITY      = 32,
   A_BIT_BIND_CHG_PROP_SECURITY,
   A_BIT_BIND_CREATE_OBJ,
   A_BIT_BIND_CREATE_PROPERTY,
   A_BIT_BIND_DELETE_OBJ,
   A_BIT_BIND_DELETE_PROPERTY,
   A_BIT_CHANGE_DATE_TIME,
   A_BIT_CHANGE_EQUIVALENCE,
   A_BIT_CHANGE_SECURITY_GROUP,
   A_BIT_UCLOSE_FILE,
   A_BIT_CLOSE_BINDERY,
   A_BIT_UCREATE_FILE,
   A_BIT_CREATE_USER,
   A_BIT_UDELETE_FILE,
   A_BIT_DELETE_USER,
   A_BIT_DIR_SPACE_RESTRICTIONS,
   A_BIT_DISABLE_ACCOUNT,
   A_BIT_DOWN_SERVER,
   A_BIT_GRANT_TRUSTEE,
   A_BIT_INTRUDER_LOCKOUT_CHANGE,
   A_BIT_LOGIN_USER,
   A_BIT_LOGIN_USER_FAILURE,
   A_BIT_LOGOUT_USER,
   A_BIT_NET_LOGIN,
   A_BIT_UMODIFY_ENTRY,
   A_BIT_OPEN_BINDERY,
   A_BIT_UOPEN_FILE,
   A_BIT_UREAD_FILE,
   A_BIT_REMOVE_TRUSTEE,
   A_BIT_URENAME_MOVE_FILE,
   A_BIT_RENAME_USER,
   A_BIT_USALVAGE_FILE,
   A_BIT_STATION_RESTRICTIONS,
   A_BIT_CHANGE_PASSWORD,
   A_BIT_TERMINATE_CONNECTION,
   A_BIT_UP_SERVER,
   A_BIT_USER_CHANGE_PASSWORD,
   A_BIT_USER_LOCKED,
   A_BIT_USER_SPACE_RESTRICTIONS,
   A_BIT_USER_UNLOCKED,
   A_BIT_VOLUME_MOUNT,
   A_BIT_VOLUME_DISMOUNT,
   A_BIT_UWRITE_FILE,
   A_BIT_GOPEN_FILE,
   A_BIT_GCLOSE_FILE,
   A_BIT_GCREATE_FILE,
   A_BIT_GDELETE_FILE,
   A_BIT_GREAD_FILE,
   A_BIT_GWRITE_FILE,
   A_BIT_GRENAME_MOVE_FILE,
   A_BIT_GRENAME_MOVE_FILE,
   A_BIT_GMODIFY_ENTRY,
   A_BIT_IOPEN_FILE,
   A_BIT_ICLOSE_FILE,
   A_BIT_ICREATE_FILE,
   A_BIT_IDELETE_FILE,
   A_BIT_IREAD_FILE,
   A_BIT_IWRITE_FILE,
   A_BIT_IRENAME_MOVE_FILE,
   A_BIT_IMODIFY_ENTRY,
   A_BIT_Q_ATTACH_SERVER,
   A_BIT_Q_CREATE,
   A_BIT_Q_CREATE_JOB,
   A_BIT_Q_DESTROY,
   A_BIT_Q_DETACH_SERVER,
   A_BIT_Q_EDIT_JOB,
   A_BIT_Q_JOB_FINISH,
   A_BIT_Q_JOB_SERVICE,
   A_BIT_Q_JOB_SERVICE_ABORT,
   A_BIT_Q_REMOVE_JOB,
   A_BIT_Q_SET_JOB_PRIORITY,
   A_BIT_Q_SET_STATUS,
   A_BIT_Q_START_JOB,
   A_BIT_Q_SWAP_RIGHTS,
   A_BIT_NLM_ADD_RECORD,
   A_BIT_NLM_ADD_ID_RECORD,
   A_BIT_CLOSE_MODIFIED_FILE,
   A_BIT_GCREATE_DIRECTORY,
   A_BIT_ICREATE_DIRECTORY,
   A_BIT_UCREATE_DIRECTORY,
   A_BIT_GDELETE_DIRECTORY,
   A_BIT_IDELETE_DIRECTORY,
   A_BIT_UDELETE_DIRECTORY
};
```

auditFileMaxSize

auditFileSizeThreshold

auditRecordCount

replicaNumber

enabledFlag

fileArchiveDays

fileArchiveHour

numOldFilesToKeep

numberReplicaEntries

auditFileCreationDateTime

reserved2

partitionID

headerChecksum

reserved3

auditDisabledCounter

auditEnabledCounter

reserved4

hdrModifiedCounter

fileResetCounter

newBitMap

reserved5

TIMESTAMP

Stores a Directory Services time stamp and holds the time stamp for the container object.

Service: Auditing

Defined In: nwaudit.h

Structure

```
typedef struct
{
   nuint32   seconds;
   nuint16   replicaNumber;
   nuint16   event;
} TIMESTAMP;
```

Pascal Structure

```
Defined in nwaudit.inc

 TIMESTAMP = Record
    seconds : nuint32;
    replicaNumber : nuint16;
    event : nuint16
  End;
```

Fields

seconds

replicaNumber

event

Client Management

Client Management: Functions

NWEndOfJob

Causes an end-of-job to be issued by the PC Shell
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, OS/2, Windows 3.1, Windows NT, Windows95
Service: Client Management

Syntax

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

NWCCODE N_API NWEndOfJob (
   void);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWEndOfJob
  : NWCCODE;
```

Return Values

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

0x0000	SUCCESSFUL

Remarks

An end-of-job is automatically issued by the PC Shell whenever a program exits unless **NWEndOfJob** has been disabled by calling the **NWSetEndOfJobStatus** function. When an end-of-job occurs, all locked files and records are cleared and any open files are closed.

NWEndOfJob can be called anytime the program needs the network environment to return to a beginning-of-program state with no files or records logged or locked.

NCP Calls

None

See Also

NWSetEndOfJobStatus

NWGetClientType

Determines the type of client running on the local workstation
NetWare Server:
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Client Management

Syntax

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

nuint16 N_API NWGetClientType (
   void);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWGetClientType
  : nuint16;
```

Return Values

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

0x0000	nuint16 with At Least One Bit Flag Set

Remarks

One or more of the following bit flags (defined in nwapidef.h) will be set when **NWGetClientType** returns:

1 NW_NETX_SHELL
2 NW_VLM_REQ
3 NW_CLIENT32
4 NW_NT_REQ
5 NW_OS2_REQ
6 NW_NLM_REQ

NWGetClientType assumes a client is loaded. If no clients are loaded, the results are inconclusive.

NCP Calls

None

NWGetRequesterVersion

Returns the major version, minor version, and revision number of the OS requester or shell

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

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

Service: Client Management

Syntax

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

NWCCODE N_API NWGetRequesterVersion (
   pnuint8   majorVer,
   pnuint8   minorVer,
   pnuint8   revision);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWGetRequesterVersion
  (majorVer : pnuint8;
   minorVer : pnuint8;
   revision : pnuint8
) : NWCCODE;
```

Parameters

majorVer

(OUT) Points to the major version number of the requester or shell.

minorVer

(OUT) Points to the minor version number of the requester or shell.

revision

(OUT) Points to the revision number of the requester or shell.

Return Values

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

0x0000	SUCCESSFUL

Remarks

If VLM.EXE is running, **NWGetRequesterVersion** returns the version of VLM.EXE, even if NETX.VLM is running.

NCP Calls

None

NWSetEndOfJobStatus

Allows an application to enable or disable the EOJs sent when COMMAND.COM executes

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

Platform: DOS, Windows 3.1

Service: Client Management

Syntax

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

NWCCODE N_API NWSetEndOfJobStatus (
   nuint8    endOfJobStatus,
   pnuint8   prevStatus);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWSetEndOfJobStatus
  (endOfJobStatus : nuint8;
   prevStatus : pnuint8
) : NWCCODE;
```

Parameters

endOfJobStatus

(IN) Specifies the end of job status.

prevStatus

(OUT) Points to the previous end of job status (optional).

Return Values

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

0x0000	SUCCESSFUL

Remarks

NWSetEndOfJobStatus is useful if the application invokes a secondary command processor and does not want to lose its files due to an end of job. The *endOfJobStatus* parameter remains set to the current setting until it is explicitly reset.

The *endOfJobStatus* parameter should be set to one of the following:

0	To disable end-of-job
1	To enable end-of-job

NCP Calls

None

Data Manipulation

Data Manipulation: Functions

bsearch

Performs a binary search of a sorted array
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
Platform: NLM
SMP Aware: Yes
Service: Data Manipulation

Syntax

```
#include <stdlib.h>

void *bsearch (
   const void   *key,
   const void   *base,
   size_t       num,
   size_t       width,
   int          (*compar) (
                       const void *,
                       const void *));
```

Parameters

key

(IN) Points to the string containing the characters to be matched.

base

(IN) Points to the sorted array.

num

(IN) Specifies the number of elements in the array.

width

(IN) Specifies the size (in bytes) of each element in the array.

compar

(IN) Points to the comparison function.

Return Values

Returns a pointer to the matching member of the array, or NULL if a matching object could not be found.

Remarks

bsearch performs a binary search of a sorted array of elements (pointed to by the *base* parameter), for an item that matches the object pointed to by the *key* parameter.

The size of each element in the array is the number of bytes specified by the *width* parameter.

The comparison function pointed to by the*compar* parameter is called with two arguments that point to elements in the array. The comparison function will return an integer less than, equal to, or greater than zero if the first argument is less than, equal to, or greater than the second parameter.

See Also

qsort

IntSwap

Reverses the 2 bytes of a short integer or short unsigned
Local Servers: nonblocking
Remote Servers: N/A
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: Yes
Service: Data Manipulation

Syntax

```
#include <nwstring.h>

WORD IntSwap (
   WORD   unswappedInteger);
```

Parameters

unswappedInteger
(IN) Specifies an integer for which high and low bytes are reversed.

Return Values

Returns the byte-swapped integer.

Remarks

An application can call **IntSwap** to reverse the high-low order of an integer being sent or received. The 80x86 family of processors store integers in low-high order. Since some of the network functions require integers to be in high-low order, integers must be converted before they are sent or received.

See Also

LongSwap

LongSwap

Reverses all 4 bytes of a long integer or long unsigned
Local Servers: nonblocking
Remote Servers: N/A
NetWare Server: 2.x, 3.x, 4.x
Platform: NLM
SMP Aware: Yes
Service: Data Manipulation

Syntax

```
#include <nwstring.h>

long LongSwap (
   long   unswappedLong);
```

Parameters

unswappedLong

(IN) Specifies the long integer for which all 4 bytes are to be reversed.

Return Values

Returns the reversed long integer.

Remarks

An application can call **LongSwap** to reverse the high-low order of a long integer being sent or received. The 80x86 family of processors store long integers in low-high order. Since some of the network functions require long integers to be in high-low order, long integers must be converted before they are sent or after they are received.

See Also

IntSwap

_lrotl

Rotates a long value to the left by the specified number of bits

Local Servers: nonblocking

Remote Servers: N/A

Platform: NLM

SMP Aware: Yes

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned long _lrotl (
   unsigned long   value,
   unsigned int    shift);
```

Parameters

value

(IN) Specifies the value to be rotated.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

_lrotl rotates the unsigned long value to the left by the number of bits specified in the *shift* parameter.

To rotate an unsigned int value, call the **_rotl** function.

See Also

_lrotr, **_rotl**, **_rotr**

_lrotr

Rotates a long value to the right by the specified number of bits

Local Servers: nonblocking

Remote Servers: N/A

Platform: NLM

SMP Aware: Yes

Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned long _lrotr (
   unsigned long   value,
   unsigned int    shift);
```

Parameters

value

(IN) Specifies the value to be rotated.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

_lrotr rotates an unsigned long value to the right by the number of bits specified in the *shift* parameter.

To rotate an unsigned int value, call the **_rotr** function.

See Also

_lrotl, **_rotl**, **_rotr**

NWLongSwap

Reverses the order of the bytes on the input long (nuint32) value
NetWare Server:
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Data Manipulation

Syntax

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

nuint32 N_API NWLongSwap (
   nuint32   swapLong);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWLongSwap
  (swapLong : nuint32
) : nuint32;
```

Parameters

swapLong

(IN) Specifies the long (nuint32) to swap.

Return Values

Returns the swapped long.

Remarks

If the original value was 0x12345678, the return value after calling **NWLongSwap** will be 0x78563412.

NCP Calls

None

See Also

NWWordSwap

NWWordSwap

Swaps the high order byte with the low order byte
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Data Manipulation

Syntax

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

nuint16 N_API NWWordSwap (
   nuint16   swapWord);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWWordSwap
  (swapWord : nuint16
) : nuint16;
```

Parameters

swapWord
(IN) Specifies the word (nuint16) to swap.

Return Values

Returns the swapped word.

NCP Calls

None

See Also

NWLongSwap

offsetof

Returns the offset of an element within a structure
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
Platform: NLM
SMP Aware: No
Service: Data Manipulation

Syntax

```
#include <stddef.h>

size_t offsetof (
   composite,
   name);
```

Parameters

composite

(IN) Specifies the structure or union for which to return an element offset.

name

(IN) Specifies the element in the structure for which to return the offset.

Return Values

Returns the offset of the element specified by the *name* parameter.

Remarks

offsetof provides a portable method to return the offset of the element specified by the *name* parameter within the structure or union specified by the *composite* parameter.

offsetof cannot be used to initialize data. It can only be used with executable statements.

qsort

Sorts an array of elements
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
Platform: NLM
SMP Aware: Yes
Service: Data Manipulation

Syntax

```
#include <stdlib.h>

void qsort (
   void    *base,
   size_t   num,
   size_t   width,
   int     (*compar) (
      const void *,
      const void *));
```

Parameters

base

(IN) Points to the array to sort.

num

(IN) Specifies the number of elements in the array.

width

(IN) Specifies the size (in bytes) of each element in the array.

compar

(IN) Points to the comparison function.

Return Values

None

Remarks

qsort sorts an array of elements (pointed to by the *base* parameter) using Hoare's Quicksort algorithm.

The size of each element in the array is the number of bytes specified by the *width* parameter.

The comparison function pointed to by the*compar* parameter is called with two arguments that point to elements in the array. The comparison function will return an integer less than, equal to, or greater than zero if the first argument is less than, equal to, or greater than the second argument.

See Also

bsearch

_rotl

Rotates an integer value to the left by the specified number of bits
Local Servers: nonblocking
Remote Servers: N/A
Platform: NLM
SMP Aware: Yes
Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned int _rotl (
   unsigned int   value,
   unsigned int   shift);
```

Parameters

value

(IN) Specifies the value to rotate.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

_rotl rotates the unsigned int value to the left by the number of bits specified in the *shift* parameter.

To rotate an unsigned long value, call the **_lrotl** function.

See Also

_lrotl, **_lrotr**, **_rotr**

_rotr

Rotates an integer value to the right by a specified number of bits
Local Servers: nonblocking
Remote Servers: N/A
Platform: NLM
SMP Aware: Yes
Service: Data Manipulation

Syntax

```
#include <stdlib.h>

unsigned int _rotr (
   unsigned int   value,
   unsigned int   shift);
```

Parameters

value

(IN) Specifies the value to rotate.

shift

(IN) Specifies the number of bits by which to rotate the value.

Return Values

Returns the rotated value.

Remarks

_rotr rotates the unsigned int value to the right by the number of bits specified in the *shift* parameter.

To rotate an unsigned long, call the **_lrotr** function.

See Also

_lrotl, **_lrotr**, **_rotl**

NDS Event

NDS Event: Guides

NDS Event: General Guide

Parent Topic:

Management Overview

NDS Event: Task Guide

Parent Topic:

NDS Event: General Guide

NDS Event: Concept Guide

- NDS Event Handling
- NDS Event Data Filtering
  - Filtering NDS Events by Local ID
    - Example: Using Local IDs to Obtain Telephone Numbers
  - Filtering NDS Events by DSTrace Events
- Global Network Monitoring
- NDS Event Functions
  - NDS Event Registration Functions
  - NDS Event Helper Functions

NDS Event: Tasks

NDS Event: Concepts

NDS Event: Examples

NDS Event: Functions

NDS Event: Structures

Parent Topic:

NDS Event: General Guide

NDS Event: Tasks

Monitoring NDS Events

The following list is a high-level view of the steps an NDS event-monitoring application must take.

1. **Register for events. Follow the steps outlined in Registering for NDS Events**
2. **Ensure that you application conforms to the following:**
    1. When the specified event occurs, such as the creation of an object, the callback is called and given data about the event.
    2. The callback determines whether to use the data. If it uses the data, the callback either immediately processes the data or makes and stores a local copy of the data. Then the callback returns.
    3. If the callback saved data locally so another thread can process it, that thread runs.
3. **Unregister for events as explained in Unregistering for NDS Events.**

Parent Topic:

NDS Event: General Guide

Processing Data Obtained by Use of Local IDs

In the example Example: Using Local IDs to Obtain Telephone Numbers, the NLM application has another thread to process the callback information in the following manner:

1. **Remove the “first in” node from the list.**
2. **Determine the name of the object whose phone number was changed.**
3. **Extract the phone number from the event data structure.**
4. **Use the object name and the phone number to update the phone number database.**
5. **Free the memory associated with node data.**
6. **Process the next node in the list.**

Parent Topic:

Example: Using Local IDs to Obtain Telephone Numbers

Registering for NDS Events

1. **Using the helper functions, determine the IDs of the desired objects, object classes, or attributes.**
2. **Call NWDSERegisterForEvent to register a function you want used as a callback when a specific event occurs. Call NWDSERegisterForEvent once for each event you want monitored.**

Parent Topic:

Monitoring NDS Events

Related Topics:

Unregistering for NDS Events

Unregistering for NDS Events

1. **When information about an event is no longer needed, call NWDSEUnRegisterForEvent to remove its callbacks from the notification lists.**

   NOTE: You must call **NWDSEUnRegisterForEvent** once for each registered event.

Parent Topic:

Monitoring NDS Events

Related Topics:

Registering for NDS Events

NDS Event: Examples

Example: NDS Event Determination

The following example shows how a callback can use the tag as parameter to determine which NDS event has occurred.

Determining Which NDS Event Has Occurred

```
int MyEventHandler(uint32 type, unsigned size, const void *data);
{
   switch(type)
   {
       case DSE_CREATE_ENTRY:
          ProcessCreateEntry(size, data);
          break;
       case DSE_DELETE_ENTRY:
          ProcessDeleteEntry(size, data);
          break;
          ...
   }
   return 0;
}
```

The same function should not be registered for multiple priorities since none of the callback's parameters specifies the priority. In addition, time constraints require different behavior from a function at one priority than from a function at another. For example, as the following discussion shows, a function registered for a given DS event at the EP_INLINE priority must complete more quickly than a function registered for the same event at EP_JOURNAL and much faster than a function registered for that event at EP_WORK.

Parent Topic:

NDS Event Types

Example: Using Local IDs to Obtain Telephone Numbers

Using Local IDs

```
/* This code would be for a callback registered with the EP_JOURNAL
   or EP_WORK priority */
/* It also assumes that the ID for the Telephone Number attribute
```

```
   has been saved in telephoneNumID. */

int PhoneNumberCallback(uint32 type, unsigned size, const void *data )
{
   const DSEValueInfo *valueInfo;

   valueInfo = data;

   if(valueInfo->attrID==telephoneNumID)
      CopyValueInfoAndSaveOnList(size, valueInfo);
   return 0;
   /*  return values are ignored for EP_JOURNAL and EP_WORK */
}
```

CopyValueInfoAndSaveOnList is a developer supplied function that would allocate space to store the data, copy the data, and then store the local copy of the data in a linked list to be processed by another thread.

> **WARNING: Your callback must not change the data pointed to by *data*. This data structure will be passed to the other event handlers, and changing it could affect the behavior of those event handlers (if they are registered).**

The following observations can be made about the above code.

The attribute ID for telephone was determined outside of the callback and stored in a global variable. This ID will not change and only needs to be determined once.

It filters only on the *attrID* field of the event data.

Rather than modifying the data, the callback makes a local copy of the data and saves the copy on a list. (Another thread will process the data.)

It returns quickly so the data can be passed to the next event handler, if there is one. See Processing Data Obtained by Use of Local IDs.

Parent Topic:

Filtering NDS Events by Local ID

NDS Event: Concepts

Filtering NDS Events by DSTrace Events

NDS Event allows NLM applications to register to the DSTrace events. These events are the same events used to report the DSTrace information when the **SET DSTRACE=ON** command is issued at the server console.

> **WARNING: Your NLM application should not rely upon the text strings supplied with the DSTrace event. These strings are for internal debugging purposes and are not guaranteed to remain the same in future OS versions.**

Parent Topic:

NDS Event Data Filtering

Related Topics:

Filtering NDS Events by Local ID

Filtering NDS Events by Local ID

When examining the data structures passed in as the data parameter of your callback, you will see that the structures use IDs rather than names. For example, the DSEValueInfo structure contains the following IDs:

perpetratorID
entryID
attrID
syntaxID
classID

While the object names in the Directory are global, the local IDs for objects on individual servers are not. Each object on a server is identified by a local ID that is relevant only on that server. The object’s local ID on another server is probably not the same.

The use of local IDs is not limited to object names. These IDs are also used to identify attributes and object classes. (The IDs for syntaxes are defined in NWDSDEFS.H.)

NDS events are reported by ID to enhance speed. IDs are 32-bit values; comparing for equality is faster with two IDs than with two strings.

In most cases, you can use these IDs as your filter.

For example, if an organization has an external telephone directory that needs to be kept current, it could create an NLM application that registers for DSE_ADD_VALUE to determine when any object's phone number changes. It would then get the attribute ID by calling **NWDSEGetLocalAttrID** and filter on the *attrID* field.

For an example of filtering NDS events by local ID, see Example: Using Local IDs to Obtain Telephone Numbers

Parent Topic:

NDS Event Data Filtering

Related Topics:

Filtering NDS Events by DSTrace Events

Global Network Monitoring

NDS Event does not provide a global solution to monitoring NDS events. Instead, it provides information that is local to each server. If your application is to provide a global solution it must do the following.

Provide an NDS event handler on each server being monitored.

Provide a method of sorting out duplicate events. For example, if there are three replicas on three servers, each having an NDS event handler registered, deleting an object will show up as a separate event on each server. In this case the global monitor must either reject all events from two of the servers or deal with receiving multiple copies of the same event.

In some implementations, it might be advantageous to obtain events from all servers that hold an instance of a partition. Such an application might be one that measures replication time in a network.

Parent Topic:

NDS Event: General Guide

NDS Event Data Filtering

When a callback is called, it must determine if the data (pointed to by the data parameter) contains information the NLM application requires. For example, if the NLM application is only concerned with changes to telephone numbers, it would use only data containing Telephone Number attribute information. Otherwise, the callback would simply return.

Data can be filtered in two ways:

By use of local IDs, as described in Filtering NDS Events by Local ID

By use of DSTraceEvents, as described in Filtering NDS Events by DSTrace Events

Parent Topic:

NDS Event: General Guide

NDS Event Functions

NDS Event provides two types of functions: registration and helper. NDS Event Registration Functions allow an NLM application to register and unregister callback functions when a specific event occurs. NDS Event Helper Functions are for accessing and evaluating the event data.

Parent Topic:

NDS Event: General Guide

NDS Event Handling

The *handler* parameter of **NWDSERegisterForEvent** points to a function called when the event occurs. Separate functions can be registered for each event or a single function can be registered for multiple events. If a callback processes multiple events, it can use the *type* parameter to determine which event has occurred.

Parent Topic:

NDS Event: General Guide

Related Topics:

Priorities of NDS Event Reporting

NDS Event Types

NDS Event Helper Functions

The functions listed in the following table are helper functions:

Name	**Description**
NWDSEConvertEntryName	Converts object names returned in the DSEEntryInfo structure to a form that is consistent with the **NWDS** functions.
NWDSEGetLocalAttrID	Retrieves the local ID of a specified NDS

	attribute.
NWDSEGetLocalAttrName	Retrieves the name of the NDS attribute associated with the supplied local ID.
NWDSEGetLocalClassID	Retrieves the local ID for the specified object class.
NWDSEGetLocalClassName	Retrieves the name of the NDS object class associated with the supplied local ID.
NWDSEGetLocalEntryID	Retrieves the local ID for the specified NDS object.
NWDSEGetLocalEntryName	Retrieves the name of the NDS object that is associated with the supplied local ID.

Parent Topic:

NDS Event Functions

NDS Event Introduction

Documentation of NDS Event provides the following kinds of information:

An explained listing of some kinds of applications that could be developed using NDS Event.

An explanation about how an NLM application can register for the NDS events.

An explanation of how to filter the event information that is reported.

An identification of issues that must be considered when creating a global monitoring application.

Parent Topic:

NDS Event: General Guide

NDS Event Priorities---EP_INLINE

EP_INLINE provides synchronous pre-event reporting:

The callback can determine whether or not the event is allowable. If the callback returns a nonzero value, the transaction is aborted, and the return value of the callback is returned to the client.

The client waits for a response while the callback processes. If the callback takes too long, the client could time out. Callbacks need to return as quickly as possible.

The callback cannot call any of the **NWDS** functions because the local database is locked. In addition, the only function it can use from NDS Event is **NWDSEConvertEntryName**.

The callback can sleep (normally only to allocate memory).

This priority faces the most difficult issues when using chained event handlers. You cannot assume that an NDS event will complete if your callback returns zero. This is because the next callback in the chain could abort the transaction. To verify changes occurred, register a callback for the EP_JOURNAL or EP_WORK priorities.

Parent Topic:

Priorities of NDS Event Reporting

Related Topics:

NDS Event Priorities---EP_JOURNAL

NDS Event Priorities---EP_WORK

NDS Event Priorities---EP_JOURNAL

EP_JOURNAL provides synchronous postevent reporting:

Event information is stores in a journal queue that records the events in the order they occurred.

A single thread services all of the callbacks for the events, so the callback’s execution time should be minimized. (The callback can determine if data should be used, and if it should, can store the data in a list that another thread processes.)

If multiple callbacks are registered for the same event, the current callback must be processed before the next callback is called.

The callback can sleep.

The callback can use any of the **NWDS** and **NWDSE** functions.

WARNING: While inside this callback, use discretion in calling NWDS functions that create more NDS events. This is a closed loop where the growth of the journal queue could be uncontrollable.

Parent Topic:

Priorities of NDS Event Reporting

Related Topics:

NDS Event Priorities---EP_INLINE

NDS Event Priorities---EP_WORK

NDS Event Priorities---EP_WORK

EP_WORK provides asynchronous postevent reporting:

The events are reported after they have occurred, but not necessarily in the order that they occurred. They are reported only after all of the event's callbacks registered for the EP_JOURNAL priority have completed.

Each callback is run on a separate thread. This frees the event handler from the time constraints of the other two priorities.

The callback can use any of the **NWDS** and **NWDSE** functions.

The callback can sleep.

Time is not a critical issue.

Parent Topic:

Priorities of NDS Event Reporting

Related Topics:

NDS Event Priorities---EP_INLINE

NDS Event Priorities---EP_JOURNAL

NDS Event Registration Functions

The functions listed in the following table are registration functions:

Name	Description
NWDSERegisterForEvent	Registers a function to be used as a callback when a specific NDS event occurs.
NWDSEUnRegisterForEvent	Unregisters a callback that has been registered to be called when a specified NDS event occurs.

Parent Topic:

NDS Event Functions

NDS Event Types

The *type* parameter of **NWDSERegisterForEvent** specifies the type of event with which to associate the callback. These types of events can be specified:

A new entry is created

An existing entry is deleted

An existing entry is renamed

An entry is moved

A value is added to an entry's attribute

A value is deleted from an entry's attribute

An attribute is deleted

A stream attribute is closed

An unused external reference is deleted

The bindery context is set on the server

A bindery object is created

A bindery object is deleted

Parent Topic:

NDS Event: General Guide

Related Topics:

Priorities of NDS Event Reporting

NDS Event Handling

NDS Event Uses

NDS Event provides a way to monitor NDS activity on an individual server. When NDS events occur, a monitoring NLM can view the events and determine what action, if any, to take. You can create applications that are notified either during or after the events have occurred.

The following list provides a few of the many uses for this service.

External Synchronization to the Directory---The Directory provides a global database that can be used to store information about organizations. External databases can use the Directory as an information source. In this case, NDS Event notification can be used to help keep the external database synchronized with the Directory.

Customized NDS Security---NDS Event notification allows a monitoring application to register to be called when specified NDS events occur. Being registered for such calls allows the callback to determine whether on not the event is allowable, thus providing you the ability to create customized NDS Security. For example, you could create an application that restricts the deletion of certain classes of objects from the Directory.

NDS Performance Analyzer---An application could watch for the replication of a specific object. The application could then time how long it takes for that object to appear on other replicas.

Parent Topic:

NDS Event: General Guide

Priorities of NDS Event Reporting

The *priority* parameter of **NWDSERegisterForEvent** specifies the registered priority of a callback. The behavior of a callback must respond partly to its registered priority, which specifies one of three kinds of reporting:

Synchronous pre-event reporting, discussed in NDS Event Priorities---EP_INLINE

Synchronous postevent reporting, discussed in NDS Event Priorities---EP_JOURNAL

Asynchronous postevent reporting, discussed in NDS Event Priorities---EP_WORK

Parent Topic:

Registering for NDS Events

NDS Event: Functions

NWDSEConvertEntryName

Converts the object names returned in the DSEEntryInfo structure to a form that is consistent with the functions whose names begin with NWDS

Local Servers: blocking

Remote Servers: N/A

Classification: 4.1*

SMP Aware: No

Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEConvertEntryName (
   NWDSContextHandle   context,
   const unicode       *DSEventName,
   char                *objectName);
```

Parameters

context

(IN) Indicates the Directory context for the request.

DSEventName

(IN) Points to the object name to be converted.

objectName

(OUT) Receives the object's name in a form consistent with the Directory Services functions.

Return Values

0x0000	SUCCESSFUL
Negative value	Negative values indicate errors. For error values, see Return Values.

Remarks

The form of the object names returned in the *dn* and *newDN* fields of the DSEEntryInfo structure is not consistent with the form used by the Directory Services functions. These names must be converted by **NWDSEConvertEntryName** before you use them with Directory

Services functions.

The format of the name returned in *newDN* is determined by the settings in the Novell Directory Services™ (NDS™) context.

The caller must allocate space for the object name to be returned. The size of the allocated memory is `((MAX_DN_CHARS)+1)*sizeof(character size)` where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode* characters are double-byte). One character is used for the NULL terminator.

NWDSEGetLocalAttrID

Retrieves the local ID of a specified NDS attribute
Local Servers: blocking
Remote Servers: N/A
Classification: 4.1*
SMP Aware: No
Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEGetLocalAttrID (
   NWDSContextHandle   context,
   const char         *name,
   uint32             *id);
```

Parameters

context

(IN) Indicates the NDS context for the request.

name

(IN) Specifies the name of the NDS attribute whose local ID is to be returned.

id

(OUT) Receives the local ID for the NDS attribute.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

An attribute's local ID is valid only for the server on which **NWDSEGetLocalAttrID** is called. For this reason, this ID is called a local ID.

The data structures returned for NDS events do not contain attribute names. Instead, these structures use local IDs to identify the attribute that is associated with the event. **NWDSEGetLocalAttrID** is used to map an attribute name (such as "User") and convert it to a local ID that can be used to compare with the local ID in an event structure.

NOTE: Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

See Also

NWDSEGetLocalAttrName

NWDSEGetLocalAttrName

Retrieves the name of the NDS attribute associated with the supplied local ID

Local Servers: blocking

Remote Servers: N/A

Classification: 4.1*

SMP Aware: No

Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEGetLocalAttrName (
   NWDSContextHandle   context,
   uint32              attrID,
   char               *name);
```

Parameters

context

(IN) Specifies the NDS context for the request.

attrID

(IN) Specifies the local ID for the schema attribute.

name

(OUT) Receives the name of the attribute associated with the local ID.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

The data structures returned for NDS events do not contain attribute names. Instead, these structures use local IDs to identify the attribute associated with the event. **NWDSEGetLocalAttrName** is used to map the local attribute ID found in the structures, to a text form of the name, such as “Telephone Number.”

NOTE: Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

The caller must allocate space for the attribute name pointed to by*name*. The size of the allocated memory is
`((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size)`
where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

See Also

NWDSEGetLocalAttrID

NWDSEGetLocalClassID

Retrieves the local ID for the specified object class

Local Servers: blocking

Remote Servers: N/A

Classification: 4.1*

SMP Aware: No

Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEGetLocalClassID (
   NWDSContextHandle   context,
   const char         *name,
   uint32             *id);
```

Parameters

context

(IN) Indicates the NDS context for the request.

name

(IN) Specifies the name of the object class whose local ID is to be returned.

id

(OUT) Receives the local class ID for the specified object class.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

An object class' local ID is valid only on the server on which **NWDSEGetLocalClassID** is called. For this reason, this ID is called a local ID.

The data structures returned for DS events do not contain object class names. Instead, these structures use local IDs to identify the object class associated with an event. **NWDSEGetLocalClassID** is used to determine the local ID for an object class, such as “User”, so the ID can be used for comparison operations.

NOTE: Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

See Also

NWDSEGetLocalClassName

NWDSEGetLocalClassName

Retrieves the name of the NDS object class associated with the supplied local ID

Local Servers: blocking

Remote Servers: N/A

Classification: 4.1*

SMP Aware: No

Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEGetLocalClassName (
   NWDSContextHandle   context,
   uint32              classID,
   unicode            *name);
```

Parameters

context

(IN) Indicates the NDS context for the request.

classID

(IN) Gives the local ID for the NDS object class.

name

(OUT) Receives the name of the object class associated with the local ID.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

The data structures returned for NDS events do not contain object class names. Instead, these structures use local IDs to identify the object class associated with an event. **NWDSEGetLocalClassName** is used to determine the name of the object class (such as “User”) that is associated with the object class ID.

> **NOTE:** Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

The caller must allocate space for the object-class name that is returned. The size of the allocated memory is `((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size)` where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

See Also

NWDSEGetLocalClassID

NWDSEGetLocalEntryID

Retrieves the local ID for the specified NDS object
Local Servers: blocking
Remote Servers: N/A
Classification: 4.1*
SMP Aware: No
Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEGetLocalEntryID (
   NWDSContextHandle   context,
   const char         *objectName,
   uint32             *id);
```

Parameters

context

(IN) Indicates the NDS context for the request.

objectName

(IN) Points to the name of the NDS object whose local ID is to be returned.

id

(OUT) Receives the local ID for the object.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

The name specified by *objectName* is relative to the context specified by *context*.

An object's local ID is valid only for the server on which **NWDSEGetLocalClassID** is called. For this reason, this ID is called a local ID.

NOTE: Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

See Also

NWDSEGetLocalEntryName

NWDSEGetLocalEntryName

Retrieves the name of the NDS object associated with the supplied local ID
Local Servers: blocking
Remote Servers: N/A
Classification: 4.1*
SMP Aware: No
Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEGetLocalEntryName (
   NWDSContextHandle   context,
   uint32              entryID,
   unicode            *objectName);
```

Parameters

context

(IN) Indicates the NDS context for the request.

entryID

(IN) Gives the local ID for the NDS object.

objectName

(OUT) Receives the name of the NDS object associated with the local ID specified by *entryID*.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

The form of the name returned by **NWDSEGetLocalEntryName** is dependant upon the settings of the flags associated with the NDS context specified by *context*.

The caller must allocate memory to receive the object name that is returned. The size of the allocated memory is `((MAX_DN_CHARS)+1)*sizeof(character size)` where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for the NULL terminator.

See Also

NWDSEGetLocalEntryID

NWDSERegisterForEvent

Registers a function to be used as a callback when a specific NDS event occurs

Local Servers: blocking

Remote Servers: N/A

Classification: 4.1*

SMP Aware: No

Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSERegisterForEvent (
   int       priority,
   uint32    type,
   int     (*handler) (
      uint32         type,
      unsigned       size,
      const void    *data);
```

Parameters

priority

(IN) Specifies the state the NDS event will be in when it is reported.

type

(IN) Specifies the type of the event for which the callback is being registered.

handler

(IN) Provides a pointer to a function to be used as a callback when the event specified by type occurs.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

priority can be one of the following values:

EP_INLINE	Synchronous pre-event reporting.
EP_JOURNAL	Synchronous post-event reporting. Events are reported after they have occurred, and in the order they occur.
EP_WORK	Asynchronous postevent reporting. Events are reported after they occur, but not necessarily in the order they occurred.

type can be one of the following values:

Event		**Event Description**
DSE_CREATE_ENTRY	1	Creation of a new NDS object
DSE_DELETE_ENTRY	2	Deletion of an existing NDS object
DSE_RENAME_ENTRY	3	Renaming of an existing NDS object
DSE_MOVE_SOURCE_ENTRY	4	This is the second of two events reported for a move operation. This event specifies the deletion of a NDS object from its original location in the Directory tree. (See DSE_MOVE_DEST_ENTRY.)
DSE_ADD_VALUE	5	Addition of a value to an object attribute
DSE_DELETE_VALUE	6	Deletion of a value from an object attribute
DSE_CLOSE_STREAM	7	Closing of a Stream attribute
DSE_DELETE_ATTRIBUTE	8	Deletion of an attribute. This generates DSE_DELETE_VALUE events for values associated with the attribute. The DSE_DELETE_VALUE events occur after the DSE_DELETE_ATTRIBUTE event.
DSE_SET_BINDERY_CONTEXT	9	Setting of the bindery context on the server
DSE_CREATE_BINDERY_OBJECT	10	Creation of a bindery object
DSE_DELETE_BINDERY_OBJECT	11	Deletion of a bindery object
DSE_MOVE_DEST_ENTRY	14	This is the first of two events reported for a move operation.

		This event specifies the placement of the NDS object into its new location in the Directory tree. (See NWDS_MOVE_SOURCE_ENTRY .) This generates DSE_ADD_VALUE events for all of the values associated with the object.
DSE_DELETE_UNUSED_EXTREF	15	Deletion of an unused external reference
DSE_TRACE	16	Occurrence of an NDS Trace event
DSE_REMOTE_SERVER_DOWN	17	
DSE_NCP_RETRY_EXPENDED	18	
DSE_REMOTE_CONN_CLEARED	19	

handler is a pointer to a function that is to be called when the specified NDS event occurs. The function is defined as follows:

type

(IN) Identifies the type of the event that has occurred. (See the*type* parameter above.

size

(IN) Specifies the size of the data that is returned for the event.

data

(IN) Points to the location of the data that contains information related to the event.

Event		**Event Description**
DSE_CREATE_ENTRY	1	DSEEntryInfo
DSE_DELETE_ENTRY	2	DSEEntryInfo
DSE_RENAME_ENTRY	3	DSEEntryInfo
DSE_MOVE_SOURCE_ENTRY	4	DSEEntryInfo
DSE_ADD_VALUE	5	DSEValueInfo
DSE_DELETE_VALUE	6	DSEValueInfo
DSE_CLOSE_STREAM	7	DSEValueInfo
DSE_DELETE_ATTRIBUTE	8	DSEValueInfo
DSE_SET_BINDERY_CONTEXT	9	No data is associated with this event.

DSE_CREATE_BINDERY_OBJECT	10	DSEBinderyObjectInfo
DSE_DELETE_BINDERY_OBJECT	11	DSEBinderyObjectInfo
DSE_MOVE_DEST_ENTRY	14	DSEEntryInfo
DSE_DELETE_UNUSED_EXTREF	15	DSEEntryInfo
DSE_TRACE	16	DSETraceInfo
DSE_REMOTE_SERVER_DOWN	17	DSENetAddress
DSE_NCP_RETRY_EXPENDED	18	DSENetAddress
DSE_REMOTE_CONN_CLEARED	19	DSENetAddress

The data structures in the above table are defined in NWDSEVNT.H.

The value returned by the callback must be 0 for success and any other value for failure. If the callback returns a nonzero value during a EP_INLINE priority event, the event will be aborted. The callback's return values for the EP_JOURNAL and EP_WORK priority events are ignored.

CAUTION: Your application must not modify the data at the location pointed to by data. Multiple callbacks can be registered for each event, and all of the callbacks receive that same data. When a callback returns, the information pointed to by data is passed into the next callback, if one is registered. Changing the information pointed to by data can produce unpredictable behavior in other callbacks. If you are going to modify the information pointed to by data, make a local copy of the information.

NOTE: The callbacks are run on threads that do not have CLIB context. If you are using functions that need CLIB context, you must establish the context by calling **SetThreadGroupID**.

See Also

NWDSEUnRegisterForEvent

NWDSEUnRegisterForEvent

Unregisters a callback that has been registered to be called when a specified NDS event occurs

Local Servers: blocking

Remote Servers: N/A

Classification: 4.1*

SMP Aware: No

Service: NDS Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

NWDSCCODE NWDSEUnRegisterForEvent (
   int      priority,
   uint32   type,
   int    (*handler)(
      uint32        type,
      unsigned      size,
      const void   *data);
```

Parameters

priority

(IN) Specifies the state for which the NDS event reporting was registered.

type

(IN) Specifies the type of the event for which the callback was registered.

handler

(IN) Provides a pointer to the function that was registered to be used as a callback when the event occurred.

Return Values

0x0000	SUCCESSFUL
Negative Value	Negative values indicate errors. For error values, see Return Values.

Remarks

For more details about the parameters for this function, see **NWDSERegisterForEvent**.

See Also

NWDSERegisterForEvent

NDS Event: Structures

DSEACL

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID;
   uint32   attrID;
   uint32   privileges;
} DSEACL;
```

Fields

entryID

Indicates the local ID for the object that received the rights.

attrID

Indicates the ID of the attribute for which the rights apply.

privileges

Indicates the effective privilege set for subject/object or subject/attribute pair.

Remarks

NOTE: The special attributes, [All Attribute Rights], [Entry Rights], and [SMS Rights] have local IDs.

Privileges are defined as follows:

All Attribute Rights

C Value	Pascal Value	Value Name
0x00000001 L	$00000001	DS_ATTR_COMPARE
0x00000002 L	$00000002	DS_ATTR_READ
0x00000004 L	$00000004	DS_ATTR_WRITE
0x00000008 L	$00000008	DS_ATTR_SELF

0x00000010 L	$00000010	DS_ATTR_SUPERVISOR

Entry Rights

C Value	**Pascal Value**	**Value Name**
0x00000001 L	$00000001	DS_ENTRY_BROWSE
0x00000002 L	$00000002	DS_ENTRY_ADD
0x00000004 L	$00000004	DS_ENTRY_DELETE
0x00000008 L	$00000008	DS_ENTRY_RENAME
0x00000010 L	$00000010	DS_ENTRY_SUPERVISOR

SMS Rights

0x00000001L	DS_SMS_SCAN
0x00000002L	DS_SMS_BACKUP
0x00000004L	DS_SMS_RESTORE
0x00000008L	DS_SMS_RENAME
0x00000010L	DS_SMS_DELETE
0x00000020L	DS_SMS_ADMIN

DSEBackLink

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   serverID;
   unit32   remoteID;
} DSEBackLink;
```

Fields

serverID

Indicates the local ID for the server that knows about the object.

remoteID

Indicates the object’s local ID on the remote server specified by *serverID*.

Remarks

The Back Link syntax is used to identify a server that knows about the object that owns the Back Link information.

DSEBinderyObjectInfo

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID;
   uint32   parentID;
   uint32   type;
   uint32   emuObjFlags;
   uint32   security;
   char     name[48];
} DBEBinderyObjectInfo;
```

Fields

entryID

Indicates the local ID for the Directory object that is being created to represent the bindery object.

parentID

Indicates the local ID for the parent of the object specified by *entryID*.

type

Indicates the bindery object type.

emuObjFlags

Indicates the bindery object flags.

security

Indicates the bindery object security.

name

Indicates the name of the bindery object.

Associated Events

DSE_CREATE_BINDERY_OBJECT
DSE_DELETE_BINDERY_OBJECT

DSEBitString

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32  numberOfBits;
   uint32  numberOfBytes;
   char    data;
} DSEBitString;
```

Fields

numberOfBits
Indicates the number of bits in the bit string.

numberOfBytes
Indicates the number of bytes in the bit string.

data
Indicates the data for the string.

Remarks

Bit strings are padded to 4-byte boundaries.

DSECIList

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   numberOfStrings;
   uint32   length1;
   unicode  string[1];
} DSECIList;
```

Fields

numberOfStrings

Indicates the number of strings the structure holds.

length1

Indicates the length (in bytes) of the first string.

string

Indicates the location of the first string.

Remarks

The stings in this structure are null terminated, and aligned on 4-byte boundaries. If necessary, the strings are padded to fit those boundaries. The value in length does not include the bytes used for padding.

The first uint32 (4 bytes) after the first string contains the length of the second string. The second string follows the length. Any remaining strings follow this pattern.

The Unicode* strings are in Intel* format, lo-hi order.

DSEEmailAddress

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   type;
   uint32   length;
   unicode  address[1 /*or more*/];
} DSEEmailAddress;
```

Fields

type

Indicates the type of the E-mail address.

length

Indicates the length of the first E-mail address.

address

Indicates the location where the first E-mail address begins.

Remarks

type can be either zero or one.

If *type* is set to zero, the*address* is an E-mail address, in the form of non-MHS_Email_Protocol:non-MHS_Email_Address. Where non_MHS_Email_Protocol is a 1-8 character string, and non-MHS_Email_Address is a string for the actual address value.

```
Example: SMTP:JohnD@Novell.Com
```

If *type* is set to one, the*address* is an E-mail alias, in the form of non-MHS_Email_Protocol:non-MHS_Email_Alias. Where non_MHS_Email_Protocol is a 1-8 character string, and non-MHS_Email_Address is a string for the actual alias value.

```
Example: SMTP:JohnD@Novell.Com
```

DSEEntryInfo

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32          perpetratorID;
   uint32          verb;
   uint32          entryID;
   uint32          parentID;
   uint32          classID;
   uint32          flags;
   DSETimeStamp    creationTime;
   const unicode  *dn;
   const unicode  *newDN;
   char            data[1];
} DSEEntryInfo;
```

Fields

perpetratorID

Indicates the local ID for the object that requested the action. For example, Admin.Wimple Makers creating an entry.

verb

Indicates the action that caused the event to occur. These verbs, such as DSV_MODIFY_ENTRY are defined in NWDSDEFS.H.

entryID

Indicates the local ID for the object that was acted upon.

parentID

Indicates the local ID for the parent of the object that was acted upon.

classID

Indicates the local ID for the object class type (such as “User”) of the object that was acted upon.

flags

Indicates the flags identifying the object type. For most object types, *flags* will be set to zero. For partition roots, external references, and aliases, *flags* will have the following values:

0x0001	DSEF_PARTITION_ROOT
0x0002	DSEF_EXTREF
0x0004	DSEF_ALIAS

creationTime

Indicates the creation time of the object that is associated with *entryID* and points to DSETimeStamp.

dn

Indicates the distinguished name of the object that was acted upon.

newDN

Indicates the new distinguished name of the object that was acted upon. This is valid only if the object's distinguished name has been changed.

data

Indicates the location where the data is stored for the *dn* and the *newDN* fields. (Do not access this data directly. Instead access it through the *dn* and *newDN* fields.)

Remarks

The distinguished names pointed to by *dn* and *newDN* are not in a form that is consistent with the names used by the NDS functions. To use these names, you must convert the names to the proper form by calling **NWDSEConvertEntryName**.

Associated Events

DSE_CREATE_ENTRY
DSE_DELETE_ENTRY
DSE_DELETE_UNUSED_EXTREF
DSE_MOVE_DEST_ENTRY
DSE_MOVE_SOURCE_ENTRY
DSE_RENAME_ENTRY

DSEFaxNumber

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32        length;
   unicode       telephoneNumber[1 /*or more*/];
   DSEBitString  parameters;
} DSEFaxNumber;
```

Fields

length

Indicates the number of characters used in the phone number.

telephoneNumber

Indicates the telephone number.

parameters

Indicates a bit string containing additional information and points to DSEBitString.

DSEHold

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32  entryID;
   uint32  amount;
} DSEHold;
```

Fields

entryID

Indicates the ID of the object that owns the accounting information.

amount

Indicates the number of charges that are on hold.

Remarks

See the Server Holds attribute in the *NDS Schema Reference* to see how this information is used.

DSENetAddress

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32  type;
   uint32  length;
   uint8   data[1];
} DSENetAddress;
```

Fields

type

Indicates the type of the address.

length

Indicates the number of bytes in which the address is stored.

data

Points to the place where the network address is stored.

Remarks

type can have the following values:

NT_IPX
NT_IP
NT_SDLC
NT_TOKENRING_ETHERNET
NT_OSI
NT_APPLETALK
NT_COUNT

The address is stored as a binary string. This string is the literal value of the address. To display it as a hexadecimal value, you must convert each 4-bit nibble to the correct character (0,1,2,3,...F).

For two net addresses to match, the type, length, and value of the addresses must match.

DSEOctetList

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32  numOfStrings;
   uint8   string1;
} DSEOctetList;
```

Fields

numOfStrings

Indicates the number of strings contained in the structure.

string1

Indicates the location of the data for the string(s).

Remarks

The strings are length preceded.

DSEPath

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32  nameSpaceType;
   uint32  volumeEntryID;
   uint32  length;
   uint8   data[1];
} DSEPath;
```

Fields

nameSpaceType

Indicates the type of the name space.

volumeEntryID

Indicates the local ID for the volume where the path is located.

length

Indicates the length of the path.

data

Indicates the location where the path is stored.

Remarks

The following name-space types have been defined:

DS_DOS
DS_MACINTOSH
DS_UNIX
FTAM
OS/2

DSEReplicaPointer

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   serverID,
   uint32   type;
   uint32   number;
   uint32   replicaRootID;
   char     referral[1];
} DSEReplicaPointer;
```

Fields

serverID

Indicates the local ID for the name server that holds the replica.

type

Indicates the replica's type.

number

Indicates the replica number.

replicaRootID

Indicates the remote ID for the object that is the partition root. This is the rpelica root's local ID on the remote server.

referral

Indicates an array of network addresses for the server specified in *serverID*.

Remarks

These types of partitions are deifned as follows:

RT_MASTER
RT_SECONDARY
RT_READONLY
RT_SUBREF

A server can have more than one address, such as IPX and IP.

DSETimeStamp

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   seconds;
   uint16   replicaNumber;
   uint16   event;
} DSETimeStamp;
```

Fields

seconds

Indicates, in seconds, when the event occurred. Zero equals 12:00, midnight, January 1, 1970, UTC.

replicaNumber

Indicates the number of the replica on which the change or event occurred.

event

Indicates an integer that further orders events occurring within the same whole-second interval.

Remarks

Two time stamp values are compared by comparing the *seconds* fields first and the *event* fields second. If the *seconds* fields are unequal, order is determined by the *seconds* field alone. If the *seconds* fields are equal, and the *eventID* fields are unequal, order is determined by the *eventID* fields. If the *seconds* and the *event* fields are equal, the time stamps are equal.

DSETraceInfo

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   unsigned long   traceVector1;
   unsigned long   traceVector2;
   unsigned long   dstime;
   unsigned long   milliseconds;
   char            string[1024];
} DSETraceInfo;
```

Fields

traceVector1

Indicates a bit flag identifying the type of trace event.

traceVector2

Is reserved.

dstime

Indicates the time when the event occurred. This is specified in whole seconds, where zero equals 12:00, midnight, January 1, 1970, UTC.

milliseconds

Indicates a further ordering (in milliseconds) of the time specified by *dstime*.

string

Indicates a string containing a message about the event.

Remarks

CAUTION: Your application should not depend upon the text strings in the DSETraceInfo sturcture. NDS™ Trace Information is for internal development purposes. The text strings returned in string may change with any version of the OS.

The bits of the *traceVector1* field are defined as follows:

Bit	Meaning
TV_ON	If set, tracing is enabled. This bit will always be set when trace events are received.

TV_AUDIT	Auditing
TV_INIT	Initialization
TV_FRAGGER	Fragger
TV_MISC	Miscellaneous
TV_RESNAME	Resolve name
TV_STREAMS	Streams
TV_LIMBER	Limber
TV_JANITOR	Janitor
TV_BACKLINK	Backlink
TV_MERGE	Merge
TV_SKULKER	Skulker
TV_LOCKING	Locking
TV_SAP	SAP
TV_SCHEMA	Schema
TV_COLL	Collisions
TV_INSPECTOR	Inspector
TV_ERRORS	Errors
TV_PART	Partition operations
TV_EMU	Bindery Emulator
TV_VCLIENT	Virtual Client
TV_AUTHEN	Authentication
TV_RECMAN	Record Manager
TV_TIMEVECTOR	Time vectors
TV_REPAIR	DS_Repair
TV_DSAgent	Low-level DSA tracing
TV_ERRET	ERRET and ERRTRACE
TV_SYNC_IN	Incoming sync traffic
TV_THREADS	DS thread scheduling
TV_MIN	Default DSTRACE messages
TV_CHECK_BIT	All TV_ values must have this bit

Associated Events

DSE_TRACE

DSETypedName

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID;
   uint32   level;
   uint32   interval;
} DSETypedName;
```

Fields

entryID
Indicates the local ID for the object.

level
Indicates the priority.

interval
Indicates the frequency of reference.

Remarks

The meaning of the information for this structure is determined by the attribute to which the information belongs.

DSEVALData

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef union
{
   unicode             string[1/*or more*/];
   uint32              num;
   uint32              entryID;
   uint32              classID;
   uint8               boolean;
   DSENetAddress       netAddress;
   DSEPath             path;
   DSEReplicaPointer   replica;
   DSEACL              acl;
   DSETimeStamp        timeStamp;
   DSEBackLink         backLink;
   DSETypedName         typedName;
   DSEHold             hold;
   DSEEmailAddress     emailAddress;
   DSEFaxNumber        faxNumber;
   DSECIList           ciList;
   uint8               octedString[1];
   DSEOctetList        octetList;
} DSEValData;
```

Fields

string

Indicates the following syntaxes:s

Case Exact String
Case Ignore String
Numeric String
Printable String
Telephone Number

num

Indicates the following syntaxes:

Counter
Integer
Interval
Time

entryID

Indicates the distinguished name.

classID

Indicates the class name.

boolean

Indicates a boolean string value.

netAddress

Points to DSENetAddress.

path

Points to DSEPath.

replica

Points to DSEReplicaPointer.

acl

Points to DSEACL.

timeStamp

Points to DSETimeStamp.

backLink

Points to DSEBackLink.

typedName

Points to DSETypedName

hold

Points to DSEHold.

emailAddress

Points to DSEEmailAddress.

faxNumber

Points to DSEFaxNumber.

ciList

Points to DSECIList.

octetString

Indicates the octet string stream.

octetList

Points to DSEOctetList.

DSEValueInfo

Service: NDS Event
Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32         perpetratorID;
   uint32         verb;
   uint32         entryID;
   uint32         attrID;
   uint32         syntaxID;
   uint32         classID;
   DSETimeStamp   timeStamp;
   unsigned       size;
   char           data[1];
} DSEValueInfo;
```

Fields

perpetratorID

Indicates the local ID for the object that requested the action. (For example, Admin.Wimple Makers added a phone number.)

verb

Indicates the action that caused the event to occur. These verbs, such as DSV_MODIFY_ENTRY are defined in NWDSDEFS.H.

entryID

Indicates the local ID for the object that was acted upon.

attrID

Indicates the local ID that identifies the type of Schema attribute that was changed.

syntaxID

Indicates the syntax that the data is stored by.

classID

Indicates the local ID that identifies the class of the object identified by *entryID*.

timeStamp

Indicates the time when the event occurred and points to DSETimeStamp.

size

Indicates the size (in bytes) of the information stored in the location identified by data.

data

Indicates the information that further identifies the changes that were made.

Remarks

The information stored in the data field of this structure is stored in a union called DSEValData.

Associated Events

DSE_ADD_VALUE
DSE_CLOSE_STREAM
DSE_DELETE_ATTRIBUTE
DSE_DELETE_VALUE

Novell Licensing

Novell Licensing: Guides

Novell Licensing: General Guide

Novell Licensing: Task Guide

Novell Licensing: Functions

Parent Topic:

Novell Licensing: General Guide

Novell Licensing: Concept Guide

NLS Status Codes

NLS Constants

NLS Executable Files

NLS DOS Executable Files

NLS Windows Executable Files

NLS NLM Executable Files

NLS Functions

License Acquisition Functions

License Unit Functions

License Certificate Functions

License System Availability Functions

License Management Functions

License Unit Availability Functions

License Certificate Management Functions

License Certificate Transaction Information Functions

License Certificate Detail Information Functions

Related Topics:

Novell Licensing: Tasks

Novell Licensing: Functions

Parent Topic:

Novell Licensing: General Guide

Installing NLS Server Software Guide

Here are detailed instructions on how to install and load NLS server software.

Tasks

How to Install NLS on NetWare 4.10 Servers

How to Install NLS on NetWare 4.11 and IntranetWare Servers

How to Install NLS on IntranetWare for Small Business Servers

How to Install NLS on Future Novell Server Releases

Concepts

NLS Load Options Overview

-l Load Option: Path/Filename

-s Load Option: Size Limit

SETUPNLS.NLM

Parent Topic:

NLS Installation

NetWare Administrator Licensing Snap-in

Licensing objects can be managed by a snap-in to NetWare Administrator. The Licensing Snap-in lets you perform administrative tasks for your licensing servers.

Using the Licensing Snap-in

Viewing the LSP Configuration

Viewing the LSP Users and Product Information

Viewing Certificate Details

Making Certificate Assignments

Viewing/Changing a Certificate Owner

Viewing Certificate Users

Installing a License Certificate File

Creating Metering Information

Parent Topic:

Novell Licensing: General Guide

Novell Licensing Introduction

This section provides an introduction to Novell Licensing.

About NLS

NLS System Components

NLS License Service Provider

NLS License Certificate Request

Client Discovery of LSP

NLS License Certificate Overview

NLS License Certificate Creation and Installation

NLS License Certificate Security

NLS License Certificate NDS Objects

NLS Certificate Assignments

Ownership of License Certificates

NLS System Requirements

Parent Topic:

Novell Licensing: General Guide

NLS Distributed Licensing

NLS is fully distributed. The distributed License Service Provider (LSP) can either work as a stand-alone server providing license services or as part of an enterprise-wide solution in concert with other LSPs. The administrator does little configuration to implement this distributed scheme, but proper planning and an understanding of the mechanisms are crucial to successful deployment.

LSP Client Requests

LSP Default Location

LSP Activation Password

Parent Topic:

Novell Licensing: General Guide

NLS Executable Files

The executable files included with this release of NLS are described below.

NLS DOS Executable Files

NLS Windows Executable Files

NLS NLM Executable Files

Parent Topic:

Novell Licensing: General Guide

NLS Installation

The following sections discuss the installation of NLS server and client components. Two installation processes are required: one for the license server and one for the client software. Each installation procedure is described below.

Installing NLS Server Software

Installing NLS Client Software

Testing Your NLS Installation

Cleaning Up After an NLS Transaction

Parent Topic:

Novell Licensing: General Guide

Testing Your NLS Installation

Here is the information you need to know in order to test your NLS installation.

NOTE: Before you run any of the test programs, you need to add a metering license to the license system.

Tasks

Adding a Metering License to the License System

Testing NLS for the DOS Client

Testing NLS for the Windows Client

Testing NLS for the 4.1x NLM Client

Checking the NLS Test Results

Concepts

Testing Your NLS Installation Overview

Parent Topic:

NLS Installation

Novell License Certificate Format Guide

This section provides an introduction to Novell Licensing.

Grant on LS_INSUFFICIENT_UNITS

Success on LS_LICENSE_TERMINATED

Grant on LS_LICENSE_UNAVAILABLE

Grant on LS_RESOURCES_UNAVAILABLE

Grant on LS_LICENSE_EXPIRED

Default Descriptor Set

Concurrent Licensing Model

Nodelocked Licensing Model (not currently available)

Personal Use Licensing Model

Site Licensing Model

One-Time Use

License Certificate Authentication and Security

LSAPI Authentication

NLS Specific Authentication

Parent Topic:

Novell Licensing: General Guide

Novell Licensing: Tasks

Adding a Metering License to the License System

Before you run any of the test programs, you need to add a metering license to the license system. To do this:

1. **Run the DOS program ADDPINFO.**

2. **Respond to the program prompts so that you create a metering license similar to this one:**

```
Publisher:NetWare LS Test Vendor
Product:NetWare LS Test Product
Version:Version 1.1
Units:2
```

Parent Topic:

Testing Your NLS Installation

Checking the NLS Test Results

1. **Check the results of your test program.**

   If the LSAPIDEM test program returns a successful status, then the installation was done properly.

   If the hex value C0001003 is returned, the client machine where the test program is run is not attached to an LSP, or the NLS is not currently running.

Parent Topic:

Testing Your NLS Installation

Cleaning Up After an NLS Transaction

If you need to delete the transaction log database:

1. **Unload the license service from the LSP by running the UNLOAD NLS and UNLOAD BTRIEVE commands at the server console prompt. The transaction database files can then be deleted.**

The location and names of these files depend on how the license service was loaded. The default file location is SYS:\SYSTEM, and the default file names are NLSTRANS.DB and NLS.IDX.

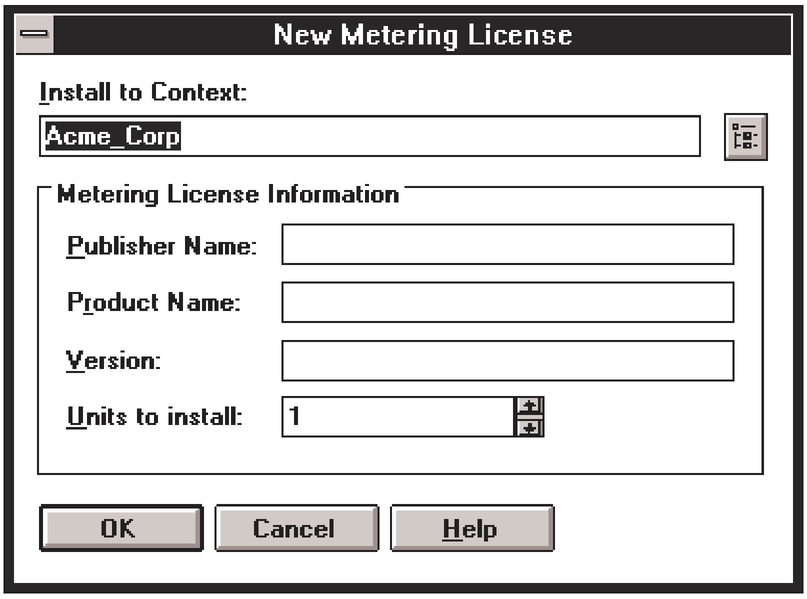
Parent Topic:

NLS Installation

Creating Metering Information

1. **Choose Install License from the Tools menu, then choose Create metering information.**

   The "New Metering License" dialog appears.



Install to Context: Acme_Corp

Units to install: 1

2. **In the Install To Context box, type the name of the user context where this certificate should be installed.**
3. **Type the Publisher Name, Product Name, and Version in the appropriate boxes.**
4. **Type or select the correct number of license units for the certificate.**
5. **When you are done, click OK to save your changes.**

   To see your changes, select Current Context from the View Menu.

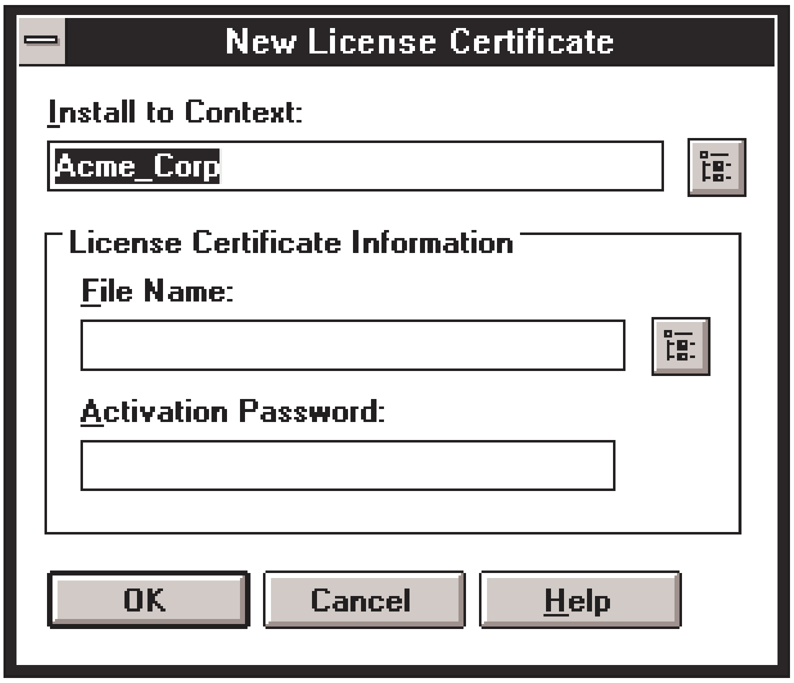
Parent Topic:

NetWare Administrator Licensing Snap-in

Installing a License Certificate File

1. **Choose Install License from the Tools menu, then choose Install license certificate file.**

   The "New License Certificate" dialog appears.



New License Certificate

Install to Context: Acme_Corp

License Certificate Information

File Name:

Activation Password:

2. **In the Install To Context box, click the Browse button and select a context where this certificate should be installed.**

3. **In the File Name box, type the filename of the license certificate.**

   You can click the Browse button to see file names.

4. **In the Activation Password box, type the activation password you are using for this license certificate, if applicable.**

5. **Click OK to install the license certificate.**

Parent Topic:

NetWare Administrator Licensing Snap-in

Installing NLS Client Software

The following instructions are for manually setting up clients to access the Novell Licensing Services. For releases later than NetWare 4.10, clients will be capable of accessing NLS as a result of the standard server installation process or after having run the server install option to "Configure Novell Licensing Services (NLS)." It is assumed that client workstations are already configured with the NetWare client software. You may copy the NLS client software to the client machine. However, be advised that this may cause incompatibilities as NLS software is upgraded on server machines.

How to Install NLS on DOS Clients

How to Install NLS on Windows 3.x Clients

How to Install NLS on Windows 95 Clients

How to Install NLS on Windows NT Clients

How to Install NLS on NLM Clients

Parent Topic:

NLS Installation

How to Install NLS on DOS Clients

A DOS TSR is required in order for DOS applications to access the Novell Licensing Services. The TSR, NLSLSAPI.EXE, is copied to the SYS:\PUBLIC directory when NLS is installed with INSTALL.NLM. The TSR passes the license API parameters from license enabled DOS applications to the LSP server.

1. **Copy NLSLSAPI.EXE to the server's SYS:\PUBLIC directory, if it is not already there. SYS:\PUBLIC is assumed to be in the client's search path.**

2. **Load the NLSLSAPI TSR manually by typing NLSLSAPI at the DOS command prompt. To load the TSR automatically in the future, place its command in the AUTOEXEC.BAT file.**

Parent Topic:

Installing NLS Client Software

NLS Installation

How to Install NLS on Windows 3.x Clients

1. **Copy LSAPI.DLL, NLSAPI.DLL and NLS.DLL to SYS:\PUBLIC. SYS:\PUBLIC is assumed to be in the client's search path.**

2. **NLSAPI.DLL provides NLS management API support to Windows 3.x clients. NLS.DLL provides LSAPI support to Windows 3.x clients. LSAPI.DLL provides indirect access to a provider's LSAPI DLL (in the case of Novell this is NLS.DLL) by first reading this information from an LSAPI.INI file. If you chose to link your product to LSAPI.DLL instead of NLS.DLL, the LSAPI.INI file is described in the LSAPI Specification v.1.1.**

Parent Topic:

Installing NLS Client Software

NLS Installation

How to Install NLS on Windows 95 Clients

1. **Copy LSAPI32.DLL, NLSAPI32.DLL and NLS32.DLL to SYS:\PUBLIC\WIN95.**

2. **NLSAPI32.DLL provides NLS management API support to Windows 95 clients. NLS32.DLL provides LSAPI support to Windows 95 clients. LSAPI32.DLL provides indirect access to a provider's LSAPI DLL (in the case of Novell this is NLS32.DLL) by first reading this information from an LSAPI.INI file. If you chose to link your product to LSAPI32.DLL instead of NLS32.DLL, the LSAPI.INI file is described in the LSAPI Specification v.1.1.**

3. **Add a search path for the SYS:\PUBLIC\WIN95 directory.**

Parent Topic:

Installing NLS Client Software

NLS Installation

How to Install NLS on Windows NT Clients

1. **Copy LSAPI32.DLL, NLSAPI32.DLL and NLS32.DLL to SYS:\PUBLIC\WIN95.**

2. **NLSAPI32.DLL provides NLS management API support to Windows NT clients. NLS32.DLL provides LSAPI support to Windows NT clients. LSAPI32.DLL provides indirect access to a provider's LSAPI DLL (in the case of Novell this is NLS32.DLL) by first reading this information from an LSAPI.INI file. If you chose to link your product**

to LSAPI32.DLL instead of NLS32.DLL, the LSAPI.INI file is described in the LSAPI Specification v.1.1.

3. **Add a search path for the SYS:\PUBLIC\WIN95 directory.**

Parent Topic:

Installing NLS Client Software

NLS Installation

How to Install NLS on NLM Clients

1. **Copy LSAPI.NLM and NLSAPI.NLM to the server's SYS:\SYSTEM directory.**
2. **Type LOAD LSAPI at the server's console command prompt.**

   LSAPI.NLM passes the common license API parameters to the LSP.
3. **Type LOAD NLSAPI at the server's console command prompt.**

   NLSAPI.NLM passes the license management API parameters to the LSP.

Parent Topic:

Installing NLS Client Software

NLS Installation

Installing NLS Server Software

The following instructions are for manually setting up servers to access the Novell Licensing Services.

How to Install NLS on NetWare 4.10 Servers

How to Install NLS on NetWare 4.11 and IntranetWare Servers

How to Install NLS on IntranetWare for Small Business Servers

How to Install NLS on Future Novell Server Releases

Parent Topic:

NLS Installation

How to Install NLS on NetWare 4.10 Servers

1. **Copy NLSLSP.NLM from the installation media to SYS:\SYSTEM on a server that will provide Licensing Services.**
2. **Copy SETUPNLS.NLM and SETUPNLS.MSG to the server.**
3. **Configure NLS by typing LOAD SETUPNLS at the server's console command prompt.**

   SETUPNLS prompts you for an admin name and password. The user you login as must have rights to the root container. SETUPNLS automatically loads NLSLSP.NLM before unloading itself.
4. **For autoloading NLSLSP.NLM in the future, add the command LOAD NLS to the server's AUTOEXEC.NCF file.**

Parent Topic:

Installing NLS Server Software

NLS Installation

How to Install NLS on NetWare 4.11 and IntranetWare Servers

1. **Type LOAD INSTALL at the server's console command prompt.**
2. **Select Product Options.**
3. **Select the option to Configure Novell Licensing Service (NLS).**

   You will need to have access to an image of the product CD ROM. NLSLSP.NLM is automatically loaded by this install process.
4. **For autoloading NLSLSP.NLM in the future, edit the server's AUTOEXEC.NCF file with a line to load the NLSLSP NLM.**

Parent Topic:

Installing NLS Server Software

NLS Installation

How to Install NLS on IntranetWare for Small Business Servers

For IntranetWare for Small Business, NLSLSP.NLM is automatically installed and loaded.

Parent Topic:

Installing NLS Server Software

NLS Installation

How to Install NLS on Future Novell Server Releases

For future releases of Novell network operating system products, NLSLSP.NLM is automatically installed and loaded.

Parent Topic:

Installing NLS Server Software

NLS Installation

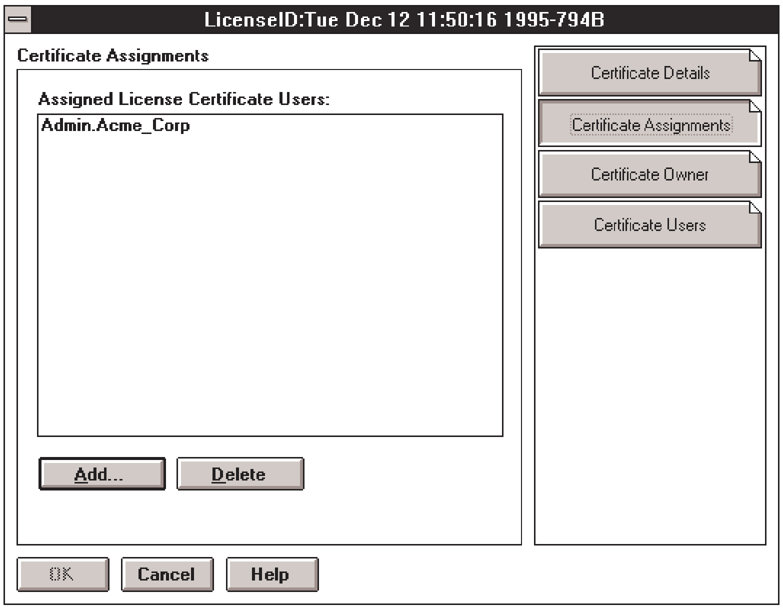
Loading NLS

1. **To see a help screen for the options that NLS can be loaded with, at the server, run LOAD NLS /?.**
2. **To set up NLS on this server the first time, run LOAD SETUPNLS and enter the network administrator name and password or equivalent user when prompted.**

Making Certificate Assignments

1. **View the certificate details for the desired license, as explained in Viewing Certificate Details.**
2. **Click the Certificate Assignment button at the right of the dialog.**

   The License Users window displays.



3. **To add new assignments, choose Add, then select one or more objects from the Select Object dialog.**
4. **To delete assignments, highlight any existing assignments to be deleted, then choose Delete.**

Parent Topic:

NetWare Administrator Licensing Snap-in

Testing NLS for the 4.1x NLM Client

1. **Copy LSAPIDEM.NLM and NLSAPIDM.NLM to a directory on the server client.**
2. **Run LOAD LSAPIDEM.**
3. **Run LOAD NLSAPIDM.**

Parent Topic:

Testing Your NLS Installation

Testing NLS for the DOS Client

1. **Copy LSAPIDEM.EXE and NLSAPIDM.EXE to a directory on the client.**

2. **Go to the DOS client and map a drive to (or log in to) a server providing the Licensing service.**

3. **In the directory where LSAPIDEM.EXE was copied, run LSAPIDEM.**

4. **In the directory where NLSAPIDM.EXE was copied, run NLSAPIDM.**

Parent Topic:

Testing Your NLS Installation

Testing NLS for the Windows Client

1. **Copy WLSAPIDM.EXE to a directory on the client.**

2. **Go to the Windows* client and map a drive to (or log in to) a server providing the Licensing service.**

3. **In the directory where WLSAPIDM.EXE was copied, run WLSAPIDEM.**

Parent Topic:

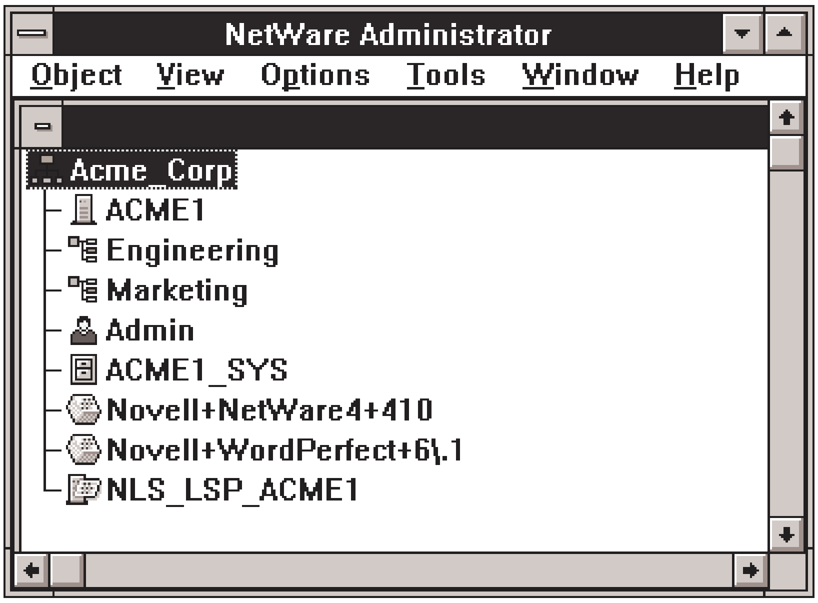
Testing Your NLS Installation

Using the Licensing Snap-in

1. **Launch the NetWare® Administrator tool from Windows*.**

2. **Choose Browse from the Tools menu to display licensing objects.**

   The example below shows sample licensing data from this screen.



The icon for an LSP looks like a computer with a license certificate. The icon for a product container looks like a stack of license certificates. The icon for a license certificate looks like a single piece of paper.

3. **To see objects beneath other objects (such as a license certificate beneath a product container), double-click on the higher-level object.**

4. **To perform a basic action, select the desired object and click the right mouse button to display an action menu (or choose an option from one of the pull-down menus).**

   The following basic options are available using the right mouse button:

Details

Rights to Other Objects

Trustees of this Object

Browse

Create

Delete

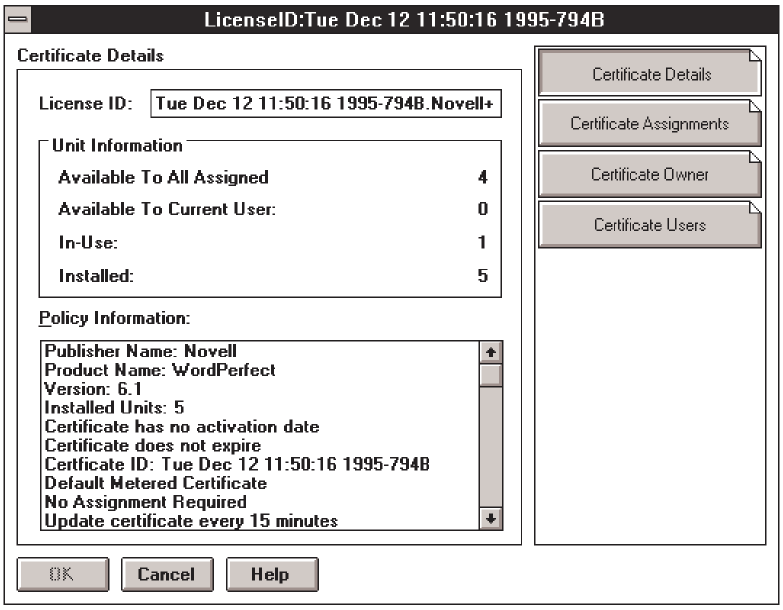
Parent Topic:

NetWare Administrator Licensing Snap-in

Viewing Certificate Details

1. **Select a license certificate, then click the right mouse button.**
2. **Choose Details from the action menu.**

   The "Certificate Details" dialog appears.


LicenseID:Tue Dec 12 11:50:16 1995-794B

Certificate Details

License ID: Tue Dec 12 11:50:16 1995-794B.Novell+

Unit Information

Available To All Assigned	4
Available To Current User:	0
In-Use:	1
Installed:	5

Policy Information:

```
Publisher Name: Novell
Product Name: WordPerfect
Version: 6.1
Installed Units: 5
Certificate has no activation date
Certificate does not expire
Certficate ID: Tue Dec 12 11:50:16 1995-794B
Default Metered Certificate
No Assignment Required
Update certificate every 15 minutes
```

The following information displays:

License ID name

Number of available license units

Number of in-use license units

Number of installed license units

Publisher's name

Product name

Version number of product

Activation date (if any)

Expiration date (if any)

Certificate ID

Whether this certificate is metered by default

Whether an assignment is required

Time interval for updating certificate

Default consumption units (begins at 1)

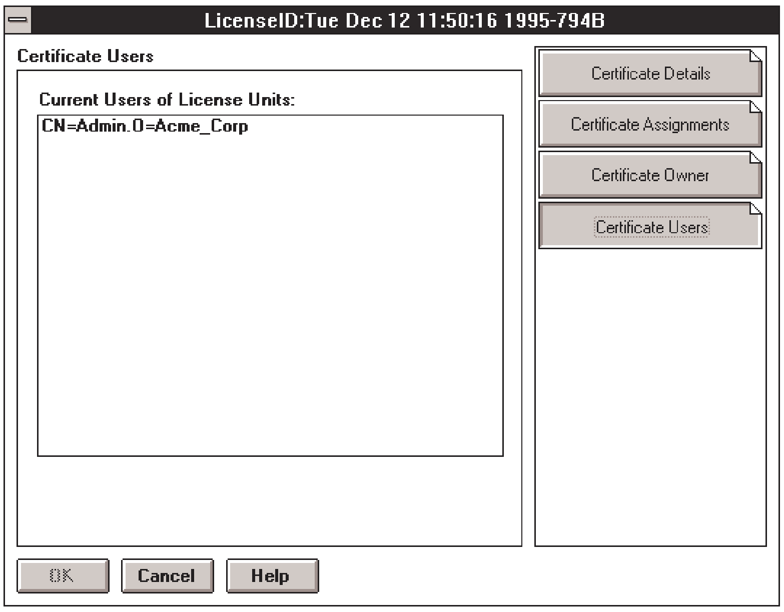
Parent Topic:

NetWare Administrator Licensing Snap-in

Viewing Certificate Users

1. **View the certificate details for the desired license, as explained in Viewing Certificate Details.**
2. **Click the Certificate User button at the right of the dialog.**

   The distinguished names of the users of this license certificate appear in the box on the screen.


LicenseID:Tue Dec 12 11:50:16 1995-794B

Certificate Users

Current Users of License Units:

CN=Admin.O=Acme_Corp

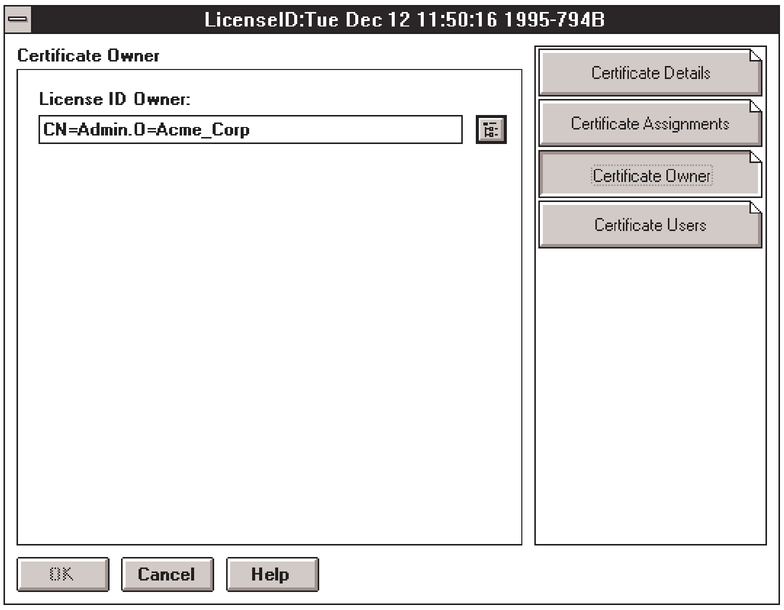
Parent Topic:

NetWare Administrator Licensing Snap-in

Viewing/Changing a Certificate Owner

1. **View the certificate details for the desired license, as explained in Viewing Certificate Details.**
2. **Click the Certificate Owner button at the right of the dialog.**

   The distinguished name of the license ID owner appears.



License ID Owner:
CN=Admin.O=Acme_Corp

3. **To search for another name to set as the owner, click the Browse button.**

Parent Topic:

NetWare Administrator Licensing Snap-in

Viewing the LSP Configuration

1. **Double-click on an LSP icon.**

   The "LSP Configuration" dialog appears, displaying the LSP name.

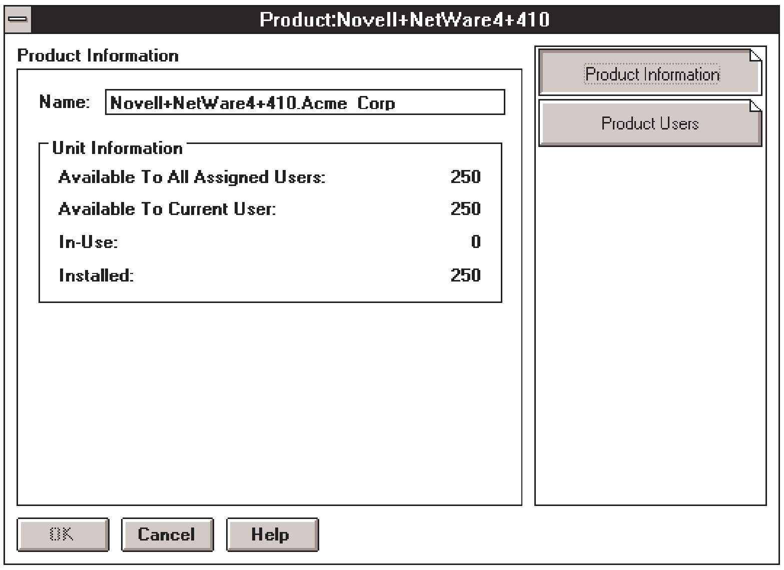
Parent Topic:

NetWare Administrator Licensing Snap-in

Viewing the LSP Users and Product Information

1. **Select a product container, then click the right mouse button.**
2. **Choose Details from the action menu.**

   The "Product Information" dialog appears.



Name: Novell+NetWare4+410.Acme_Corp

Unit Information

Available To All Assigned Users:	250
Available To Current User:	250
In-Use:	0
Installed:	250

   The following Unit Information also appears:

   Number of products available to all assigned users

   Number of products available to current user

   Number of products currently in use

   Number of installed products

3. **To view current users of the product, click the Product Users button at the right of the dialog.**

   This shows all users who are using license certificates of this type, from this container and above.

Parent Topic:

NetWare Administrator Licensing Snap-in

Novell Licensing: Concepts

About NLS

The Novell Licensing Service (NLS) is a distributed, network service that helps administrators monitor and control the use of licensed applications on a network. NLS enables a flexible approach to licensing on the NetWare 4.1x network operating system. NLS is tightly integrated with NDS and therefore benefits from NDS with fault tolerance, scaleability, and manageability of license objects

This architecture consists of client components that support different platforms and a system component called the license service provider (LSP) that runs on a NetWare 4.1x server. NLS supports DOS, Windows 3.1, Windows 95, Windows NT and NetWare 4.1x NLM clients. NLS also provides a NetWare Administrator license management snap-in tool, as well as libraries that export licensing functionality to developers.

NLS provides a foundation for software vendors to sell their software using electronic licenses instead of paper ones. Since policy information is contained in the license, only one executable image is necessary. With a variety of policy attributes to choose from, flexible license stratifications can be created for the same executable. This means that a trial copy and a permanent copy of a software product will use the same shipping executable but will have different licenses.

NLS activity is stored in transaction databases.   Management APIs provide access to these transaction databases. Therefore application usage on the network can be monitored.

NLS is based on the industry-standard LSAPI Specification v1.1. If an application uses these APIs and the license conforms to the specification, any LSP based on this specification such as NLS could service the application.

Parent Topic:

Novell Licensing Introduction

License Acquisition Functions

The NLS Client uses the standard License Services Application Programming Interface (LSAPI v1.1) to obtain and maintain license units from a particular publisher for a specific product and version. These API functions allow the application provider to license applications independent of an underlying Licensing service.

The License Acquisition functions help you do the following things:

Determine what Licensing services are available.

Request license units for a particular product.

Tell the Licensing service that the previously requested license units are still in use.

Release license units previously granted by the Licensing service.

Determine how often a license certificate needs to be updated.

Translate an error code to a displayable text string.

Parent Topic:

Novell Licensing: General Guide

Related Topics:

License Unit Functions

License Certificate Functions

License System Availability Functions

License Certificate Detail Information Functions

NLSCertificateTagToMessage	Presents license certificate information to the user in a more clear and organized manner.
NLSGetCertificate	Retrieves the data that makes up a license certificate.

Parent Topic:

License Management Functions

Related Topics:

License Unit Availability Functions

License Certificate Management Functions

License Certificate Transaction Information Functions

License Certificate Functions

LSQuery	Gets information about a license certificate.
LSGetMessage	Gets a string describing the current error code.

Parent Topic:

License Acquisition Functions

Related Topics:

License Unit Functions

License System Availability Functions

License Certificate Management Functions

NLSInstallCertificate	Installs a license certificate into the specified licensing system.
NLSDeleteCertificate	Removes a license certificate record from the license database.
NLSMoveCertificate	Relocates the database record containing the specified license certificate.
NLSAddProductInformation	Adds non-secure license certificate information to the licensing system database.
NLSTransferOwnership	Transfers the ownership of a license certificate.
NLSAddAssignment	Restricts the use of a license certificate based on assignment type.
NLSRemoveAssignment	Removes an assignment from a license certificate.

Parent Topic:

License Management Functions

Related Topics:

License Unit Availability Functions

License Certificate Transaction Information Functions

License Certificate Detail Information Functions

License Certificate Transaction Information Functions

NLSGetEntry	Views all the individual components of a transaction.
NLSGetTransacti on	Retrieves a single transaction record.

Parent Topic:

License Management Functions

Related Topics:

License Unit Availability Functions

License Certificate Management Functions

License Certificate Detail Information Functions

License Management Functions

The License Management functions help determine the status of the Licensing service and assist in configuring and maintaining the Licensing service. For example, these functions can be used by an administrative tool or a third-party system to manage software assets. They provide access to license usage, license restrictions, license installation and information, and transactional records.

The License Management functions accomplish the following tasks:

Determine how many license certificates and units are installed, how many are in use, how many are available, and who is using them.

Add or remove license assignments.

Determine or transfer ownership of a license certificate.

Install a license certificate into the system.

Remove, move, or examine installed license certificates.

Access a transaction database to determine history of use or errors.

Parent Topic:

Novell Licensing: General Guide

Related Topics:

License Unit Availability Functions

License Certificate Management Functions

License Certificate Transaction Information Functions

License Certificate Detail Information Functions

License System Availability Functions

Table auto. License System Availability Function

LSEnumProviders	Returns a unique string containing the name of any accessible licensing system.

Parent Topic:

License Acquisition Functions

Related Topics:

License Unit Functions

License Certificate Functions

License Unit Availability Functions

NLSInstalled	Provides information about license units that are installed for products in one or more licensing systems.
NLSAvailable	Provides information about license units available for products in one or more licensing systems.
NLSInUse	Provides information about license units that are in use for products in one or more licensing systems.
NLSUsers	Provides information about who or what is using licensing units for a product in the specified licensing systems.

Parent Topic:

License Management Functions

Related Topics:

License Certificate Management Functions

License Certificate Transaction Information Functions

License Certificate Detail Information Functions

License Unit Functions

LSRequest	Requests licensing resources associated with a specific software product.
LSUpdate	Verifies that the current handle is still valid.
LSRelease	Releases licensing resources associated with the license.
LSFreeHandle	Releases all resources associated with the specified licensing context.

Parent Topic:

License Acquisition Functions

Related Topics:

License Certificate Functions

License System Availability Functions

LSP Activation Password

An activation password is required when using the LSAPI secrets security attribute. If the LSAPI secrets attribute is not utilized, a blank password should be entered at license installation time (using **NLSInstallCertificate**).

NOTE: The activation password must be entered, exactly as it appears, using the NetWare® Administrator Licensing Snap-in or your own license installation process. If you are using the example license generation utility, the activation password is currently hard-coded as 1234567890123456. If you change the activation password, it should be exactly 16 characters long to ensure proper entry and security. There are no other restrictions on the content of this password. It is suggested that the password value be an MD4 message digest of your publisher name, a randomly chosen string, and a time stamp value. This provides a unique activation password of the proper length and format for all your generated license certificates. Using the same activation password for all license certificates significantly reduces the security provided by this attribute, and is not recommended.

Parent Topic:

NLS Distributed Licensing

LSP Client Requests

The LSP attempts to process a client request by checking the client's current container first. If the necessary information is not available within the same NDS container, the LSP searches the next container "up the tree" for the requested information. ("Up" refers to searching the next higher (closer to the root) level container within the tree.) This process continues until either the necessary information is retrieved to complete the client request, or until the root container has been reached and searched. This search process allows for the rapid placement of license certificates in Organizations or Organizational Units. A cross-container search does not occur for license certificate objects. Therefore, placing a certificate into a container inherently restricts access to that certificate to users in the given container and any of its sub-trees.

NOTE: Aliases could be created within different paths in the overall tree, which represent the same physical instance of a license certificate. This allows access through multiple paths to the same set of license certificates, overriding the default cross-container isolation.

Parent Topic:

NLS Distributed Licensing

LSP Default Location

When NLS is set up on a server, the schema is extended and an LSP object is created in the same container as the NCP object (NetWare® Server) on which it resides. This is the default location for a given LSP.

Parent Topic:

NLS Distributed Licensing

NLS Certificate Assignments

A license certificate assignment specifies an NDS object which may use the license units in the certificate. When a certificate is first installed it has no assignments. A license certificate with no assignments will allow any NDS object, in the current container or below it, to use license units from the license certificate. Making an assignment to a certificate limits the NDS objects which may use the license units. Examples of a DN which may be assigned to a license certificate include a user object, a group object, or a container object. Assignment is made by using the NLSAddAssignment function. Once an assignment is made to a license certificate an NDS "Security Equal To" is performed on the connection requesting the object.

Parent Topic:

Novell Licensing Introduction

NLS Constants

`LS_NULL`	Empty string.
`LS_ANY`	Use any valid value for license system, product, version, and so on.
`LS_DEFAULT_UNITS`	Licensing should determine the appropriate value for "total units consumed," using its own licensing policy mechanisms.
`LS_INFO_NONE`	Reserved
`LS_INFO_DATA`	Returns a block of vendor-defined application data from the license certificate. Allocated space for such data varies from system to system or might not be available at all. The first LS_ULONG in the data buffer indicates the size in bytes of the data that follows.
`LS_INFO_SYSTEM`	Returns the unique ID of Licensing supplying the current license context. This ID is a NULL-terminated string equivalent to the value returned by **LSEnumProviders**.
`LS_MACHINE`	Restricts based on a machine address. This allows a license certificate to be assigned to one or more machines.
`LS_UPDATE_PERIOD`	Returns the recommended interval in minutes at which **LSUpdate** should be called. This value is returned as an LS_ULONG in the data buffer. If a value of 0xFFFFFFFF is returned, no recommended update period exists for the associated system.
`LS_USER_NAME`	Restricts based on a user or object name. NDS containers or group names can be specified. All elements that exist under a container or group then have access.

Parent Topic:

Novell Licensing: General Guide

NLS DOS Executable Files

In combination, the static library LSAPIx.LIB and the TSR NLSLSAPI.EXE provide license acquisition access to NLS for DOS clients.

LSAPIx.LIB	LSAPIx.LIB is statically linked to the DOS application that requires license resources and enables dynamic communications with one or more LSAPI-compliant TSRs. The **x** represents the model identifier for the application's memory model.
NLSLSAPI.EXE	NLSLSAPI.EXE is an LSAPI-compliant TSR for NetWare® Licensing. The TSR must be loaded before the DOS client application is executed; otherwise, the `LS_SYSTEM_UNAVAILABLE` error is returned for any LSAPI function.
NLSAPIx.LIB	These model-specific libraries provide access to the NLS management functions.

Parent Topic:

NLS Executable Files

NLS License Service Provider

The License Service Provider (LSP) is the main engine for NLS. LSP functionality is provided by NLSLSP.NLM. An installation of NLS requires at least one LSP to service clients.

The LSP provides the following service for a client:

1. Access by an application (client) to a license represented in NLS by a license certificate. This access is for applications that use the license service to control or monitor usage. The application uses APIs to send license requests to a license service provider (LSP) which then reads the NLS license certificate and returns the appropriate behavior for the application. The License Acquisition Functions and theNovell License Certificate Format conform to the industry-standard LSAPI Specification v1.1. The license certificate isintroduced in the NLS License Certificate Overview.

2. Access to administrative or management functions as described in the License Certificate Management Functions. Functionality includes the ability to install, move, and make assignments to grant access to license certificates as well as to read transaction databases.

LSAPI specification functions are prefixed with LS. Management functions are prefixed with NLS.

Parent Topic:

Novell Licensing Introduction

NLS License Certificate Request

NLS uses the following steps to process a license certificate request for license units:

1. A client-specific component packages the license request and submits it to the LSP.
2. The LSP examines the license request and determines if a license with sufficient units is available. It does this by first checking the client NDS context for license certificates.
3. If the LSP is able to fulfill the license request, it allocates license units for use by the client.
4. If the LSP cannot fulfill the request, it checks each next-higher context in NDS to find the necessary license units. This process continues until the necessary license units are obtained or until the LSP can search no further.
5. License request activity is recorded in a transaction database for further analysis using License Management Functions

Parent Topic:

Novell Licensing Introduction

Client Discovery of LSP

An NLS enabled client is any application that requests NLS services. The NLS requests are made through a local licensing system software component such as a static library/TSR combination or a set of dynamic libraries. NLS supports DOS, Windows 3.1, Windows 95, Windows NT, and Novell 4.1x NLM clients.

The NLS client discovers the license service provider (LSP) by checking the current connection to see if it is attached to an LSP. If not it checks for an LSP object in the tree and then attaches to those servers listed in the object.

For the DOS, Windows 3.1, Windows 95, and Windows NT platforms, the client discovers the license service provider (LSP) by checking the preferred connection. If the connection is not attached to an LSP server, the client then checks the rest of the connections in the connection table. If none of the servers that the client connected to are and LSP, then the NLS client searches NDS in the current user context for an LSP object. If an LSP object is found, the client attaches to an LSP server listed in the LSP object. If an LSP object is not found in the user context, the client checks the next-higher NDS context for LSP objects. This process continues until an LSP object is found or until the LSP can search no further.

For an NLM client the process is basically the same. The difference is that the connection table is not searched. Only the current connection is checked to see if it is connected to an LSP server before a search for an LSP object begins.

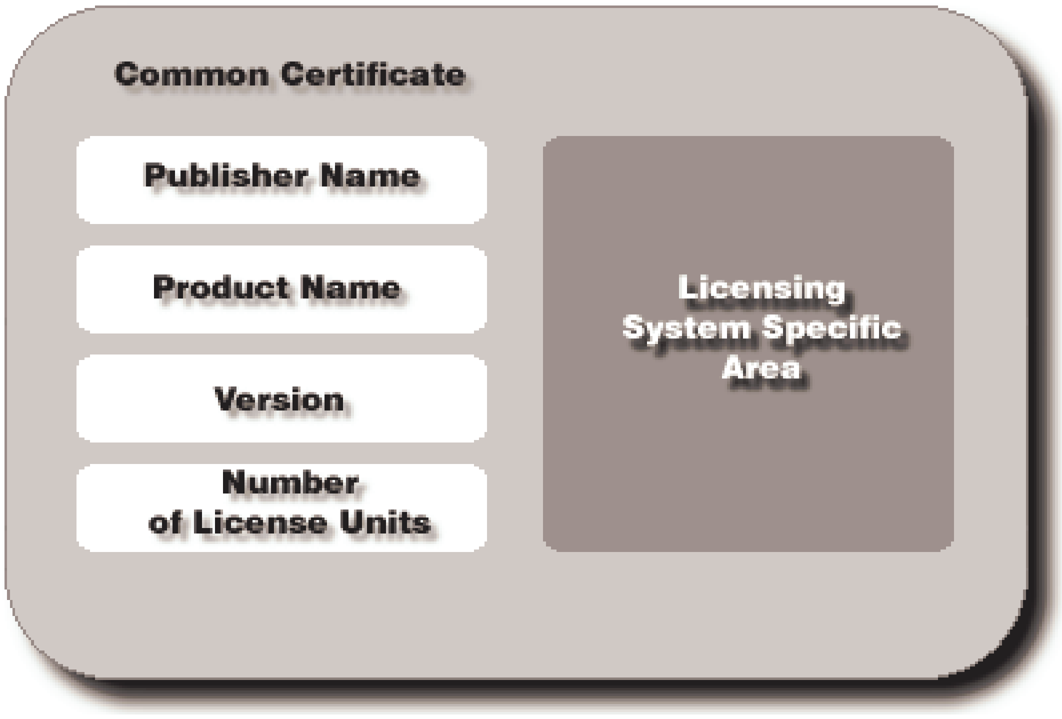
Parent Topic:

Novell Licensing Introduction

NLS License Certificate Overview

An NLS license certificate is composed of a series of license attributes. These license attributes give the developer the flexibility to create a variety of different behavior policies. License attributes are nested/appended together in order to form a certificate. In addition to required license attributes in the common certificate area a Novell License Certificate defines optional license attributes known as policy descriptors which can be placed in the license system specific area of the certificate. These policy descriptors include consumption, security, unit related, and error handling attributes.

Figure 1. NLS License Attributes



The following example demonstrates the flexibility of execution which can be obtained when creating a license certificate:

Application X is license enabled using LSAPI specification APIs described in the License Acquisition Functions. This application is shipped with different beta licenses for different groups of beta testers. One license certificate offers 5 units and expires July 1, 1998. The other license certificate is for use by a preferred set of testers. It is created with the same number of units and the same expiration date of July 1, 1998, but it establishes an exception policy that allows an additional 30 days of usage beyond the July 1 expiration. If the second license is used during the exception period, this usage will be recorded in the transaction database for further analysis.

A license certificate may be installed, moved, or deleted. However, for the license certificate to remain credible and secure, information contained in the license certificate cannot be modified.

NLS security mechanisms which are described in License Certificate Authentication and Security are based on the use of specific license certificate attributes. Details about the license certificate format are found in the Novell License Certificate Format.

Parent Topic:

Novell Licensing Introduction

NLS License Certificate Creation and Installation

A license certificate is classified as secure or non-secure. A non-secure license, also known as a metering license, is used primarily to monitor application usage. A metering license can be created and installed programmatically with the **NLSAddProductInformation** function or by using the create metering information... pull-down option from the NetWare Administrator Licensing Snap-in. You will be required to supply the required license attributes of Publisher Name, Product Name, Version, and Number of Units.

Secure license certificates are created by using a license generation utility. Currently, Novell does not support an official license generation utility but this SDK provides the LICGEN example along with the TMPLTOOL license generation template which demonstrates how such a utility could be constructed.

After the license certificate is created it can be installed using the **NLSInstallCertificate** function or by using the install license certificate file... pull-down option from the NetWare Administrator Licensing Snap-in.

Parent Topic:

Novell Licensing Introduction

NLS License Certificate Security

Secure licenses support two levels of security. The first level of security is defined by the LSAPI v1.1 Specification, Appendix B: Basic Challenge and described in LSAPI Authentication of that same document. License creation, installation, and application execution requires four 32 bit secrets (LSAPI secrets), an activation key, and challenge/response code. The LSAPIDEM example on the SDK contains this challenge/response code which can be used in your application. Details of creating an activation key and encrypting/decrypting secrets can be found in LSAPI Authentication. When LSAPI Secrets are used, an activation key will be required when installing the certificate.

The second level of security which is supported by NLS utilizes vendor defined encryption such as the use of a public/private key pair. Details can be found in NLS Specific Authentication.

Parent Topic:

Novell Licensing Introduction

NLS License Certificate NDS Objects

In addition to the license certificate itself, two types of objects are stored in NDS relating to licenses; a license certificate object and a product container object. The license certificate object contains information about license certificate required attributes as well as the license certificate owner, certificate assignments if any, and DNs of any object currently using units of the license.

The product container object is used to organize multiple license certificates of the same type under a common object. For example, a user may have four license certificates for IntranetWare for Small Business. The four license certificates have the same publisher, product, and version attributes so they are stored under the same product container. In addition, the product container object contains runtime summary data.

Parent Topic:

Novell Licensing Introduction

NLS Load Options Overview

The correct useage of the NLS load command line is:

```
LOAD NLSLSP [-t] [-l <path/file.] [-s <size>] [/?]
```

The options are briefly described below.

Option	Description
-l<*path/filename*>	Specify a location/name for the transaction database.
-s<*size in KB*>	Specify the maximum size of the transaction database.
-t<+/->	Turns transaction logging on (+) or off (-). The default is on (+).
/?	Help screen listing the usage.

You can not combine options as follows:

```
LOAD NLSLSP -t
```

```
LOAD NLSLSP -t+
```

```
LOAD NLSLSP -t-
```

Parent Topic:

Installing NLS Server Software

Related Topics:

-l Load Option: Path/Filename

-s Load Option: Size Limit

NLS NLM Executable Files

LSAPI.NLM, LSAPI.IMP	The LSAPI.NLM shared library NLM provides the LSAPI v1.1 calls for NetWare® NLM clients. This module should be loaded before the invocation of the licensed NLM, either manually or through a module directive. The LSAPI.IMP file contains all of the exported function names in the LSAPI.NLM module.
NLSAPI.NLM, NLSAPI.IMP	The NLSAPI.NLM shared-library NLM provides the access to the NLS management functions. This module should be loaded before the invocation of the licensed NLM, either manually or through a module directive. The NLSAPI.IMP file contains all of the exported function names in the NLSAPI.NLM module.
NLSLSP.NLM	NLSLSP.NLM is the NLS LSP. You can load LSPs on servers throughout your directory tree.

LM	servers throughout your directory tree.

Parent Topic:

NLS Executable Files

NLS Status Codes

`LS_SUCCESS`	The requested function completed successfully.
`LS_AUTHORIZATION_UNAVAILABLE`	The license certificate or specified assignment was not located, or the required security equivalence to the owner was not valid.
`LS_BAD_ARG`	One or more parameters are invalid.
`LS_BAD_HANDLE`	The specified handle does not describe an existing transaction handle.
`LS_BAD_INDEX`	The specified index does not point to a valid license certificate. This usually occurs when the end of the requested information has been completely returned.
`LS_BUFFER_TOO_SMALL`	The buffer to receive output was not large enough to hold the requested output.
`LS_INSUFFICIENT_UNITS`	Licensing could not find enough installed license units to satisfy the request at this time.
`LS_LICENSE_EXPIRED`	Re-verification of the existing units failed due to an out-of-date license certificate.
`LS_LICENSE_TERMINATED`	Re-verification of the existing units failed, so no additional units were requested. An update was not done within the specified update period, and the license units were issued to another user by NLS before this update was attempted.
`LS_LICENSE_UNAVAILABLE`	Re-verification of the existing license units was successful, but there were not enough available license units to fulfill the additional request.
`LS_RESOURCES_UNAVAILAB`	Insufficient resources (such as

`LE`	memory) available for request.
`LS_SYSTEM_UNAVAILABLE`	DOS TSR or Windows* DLL is not properly configured or available, or client has no Licensing service to communicate with.
`LS_TEXT_UNAVAILABLE`	Unknown status code was passed to this routine for translation.

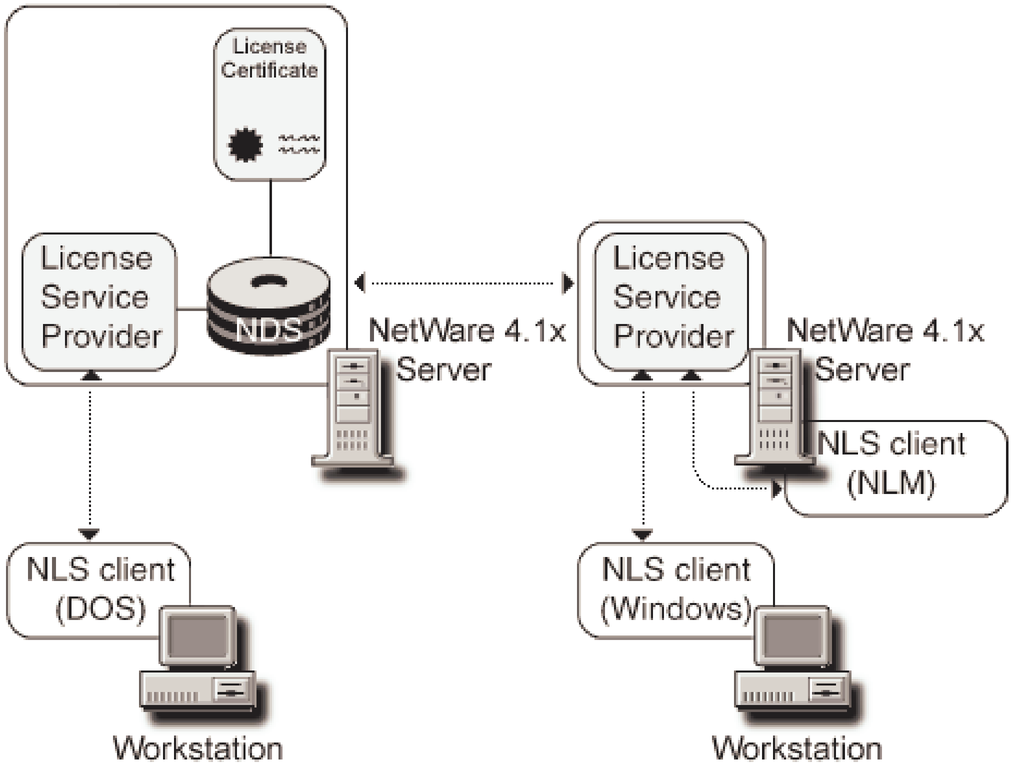
Parent Topic:

Novell Licensing: General Guide

NLS System Components

As indicated in the following figure, the Novell Licensing Service consists of client components and server components that reside on NetWare 4.1x servers.

Figure 2. NLS Client and System Components



The following components comprise the Novell Licensing Service:

One or more License Service Providers (LSPs).

Transaction databases (not shown above). A separate transaction database resides on each server running an LSP (NLSLSP.NLM) and contains application (client) activity.

Platform-specific client components.

Licenses.

Parent Topic:

Novell Licensing Introduction

NLS System Requirements

NLS has been tightly integrated with NDS to take advantage of the latter's features. Therefore, the NLS server software **must** be run on a NetWare® 4.1x server. The client platforms supported are MS-DOS 3.3 and above, Windows* 3.1 and Windows95, and NetWare 4.1.

Client/Server Type	Requirements
License Service Provider	NetWare 4.1x server with 10 MB of memory (minimum); NDS installed on server. Standard NLM applications (CLIB, STREAMS, BTRIEVE, AFTER311, DSAPI) exist on the server. Any volume with at least one megabyte of disk space can be used.
DOS Client	MS-DOS 3.3 or above; 500 KB memory (minimum); 250 KB disk space
Windows Client	Windows 3.1; 100 KB disk space or Windows95; 100 KB disk space
NLM Client	NetWare 4.1x server

Parent Topic:

Novell Licensing Introduction

NLS Windows Executable Files

These files might be necessary to enable license acquisition access to NLS from Windows* clients.

from Windows* clients.

LSAPI.LIB, LSAPI.DLL	LSAPI.LIB and LSAPI.DLL are the generic LSAPI v1.1 client components for accessing any Licensing service independently. LSAPI.LIB is an import library for the generic LSAPI.DLL. The LSAPI.DLL reads the required, local LICENSE.INI file and loads the appropriate client components. For more information on this process, see the description of NLS.LIB and NLS.DLL below.
NLS.LIB, NLS.DLL, NLS32.DLL	These are the NLS-specific client components that enable license acquisition access to NLS from Windows clients. Normally, a Windows application that requires license resources is compiled with the generic LSAPI.LIB, an import library. LSAPI.LIB lets Windows application access any Licensing service via the generic LSAPI.DLL, which loads NLS.DLL (or NLS32.DLL) after reading the LICENSE.INI file. If you are developing a Windows client specifically for NLS, then a direct link can be developed with NLS.LIB, the NLS-specific import library. This library communicates with NLS.DLL directly. An application developed in this manner does not use LSAPI.LIB, LSAPI.DLL, or LICENSE.INI and is not licensing system-independent.
NLSAPI.DLL, NLSAPI32.LIB, NLSAPI32.DLL	These are the licensing service management APIs.

Parent Topic:

NLS Executable Files

Ownership of License Certificates

The owner of a license certificate is the NDS object which can manage the license certificate; license certificate management includes making license certificate assignments, moving license certificates, and deleting license certificates.

The license certificate owner can be a specific user, a group, or any other NDS object. To allow an object to manage a license certificate, simply make that object security-equivalent to the Distinguished Name (DN) identified in the Ownership attribute or change the ownership to that DN.

Initially, the Ownership attribute is filled in with the DN of the NDS object that installed the license certificate. If necessary, that object can transfer ownership of the certificate to another object with NLSTransferOwnership. For example, you can transfer ownership to a group or role object to allow different approaches to license certificate management.

Parent Topic:

Novell Licensing Introduction

SETUPNLS.NLM

SETUPNLS.NLM is a utility used to configure a server to provide licensing services. You should only need to run SETUPNLS when setting up a NetWare 4.10 server to run NLS. For NetWare 4.11 and IntranetWare, the administrator may use SETUPNLS, but using INSTALL to configure a server for licensing services will do the same thing and is the preferred method. If the administrator chooses to use INSTALL, SETUPNLS is actually spawned by the Configure NetWare Licensing Service option in the Product Option of the INSTALL NLM. For IntranetWare for Small Business and for future releases of Novell network operating system products, NLS is installed and configured automatically during the standard installation process.

SETUPNLS, whether run by itself or via INSTALL, requires a login by an administrator who has rights to make changes to the NDS tree. SETUPNLS extends the NDS schema and creates a License Service Provider (LSP) object for the server on which it is being run. It also creates a License Servers group at the organization level in the NDS tree and adds the server to this group. Finally, SETUPNLS loads NLSLSP.NLM, and then is unloaded. SETUPNLS does not modify AUTOEXEC.NCF to load NLSLSP.NLM. For NLS to load automatically each time the server is started, add the line LOAD NLSLSP to the AUTOEXEC.NCF file in SYS:\SYSTEM. Otherwise, manually load NLSLSP.NLM.

For additional information about installing and configuring NLS, see NLS Installation.

Parent Topic:

Installing NLS Server Software

Testing Your NLS Installation Overview

The following test programs come with NLS. They each test whether the licensing service and the client workstations have been set up correctly:

LSAPI test programs:

LSAPIDEM.EXE (for DOS)

WLSAPIDM.EXE (for Windows 3.1)

LSAPIDEM.NLM (for 4.1x NLM)

NLSAPI test programs:

NLSAPIDM.EXE (for DOS)

NLSAPIDM.NLM (for 4.1x NLM)

When one of the LSAPI test programs is run, it attempts to show the available license systems, requests a test license, updates the license certificate, and releases it.

When one of the NLSAPI test programs is run, it shows the following things:

Number of and details of installed license certificates

License units in use

License units available

Users using license units

Transaction log

Parent Topic:

Testing Your NLS Installation

-l Load Option: Path/Filename

The transaction database holds a record of all modifications done to the licenses. By default, the path of the transaction database is SYS:\SYSTEM\NLSTRANS.DB. To specify a different location for the transaction database, add `-l <location/filename>` to the command line. (To use the default location for the transaction database do not use the -l option.) For, example, to put the transaction database on a volume called `FILES:` and in a directory called `database`, you would type `LOAD NLS -l FILES:\DATABASE\NLSTRANS.DB`.

The -l option must have a full volume, directory, filename specification. The volume and directory must exist, or NLSLSP.NLM is loaded incorrectly.

Parent Topic:

NLS Load Options Overview

-s Load Option: Size Limit

The size option limits the size of the transaction database; this is helpful in conserving disk space. When the -s option is used and a change is made to the transaction database that would put its size over the limit, the oldest transactions recorded in the database are removed to make room for the new transactions to be recorded. This way the transaction database always has the most current information about what has been happening with the license certificates. For example, to load NLS and set the transaction database size to about 20 KB, enter this at the console command prompt:

```
LOAD NLSLSP  -s  20
```

If the license server is ever downed and then brought back up, this option should not be set to a size smaller than the current transaction database size. Each transaction is kept as a whole unit, so this can affect the actual size of the transaction database.

Parent Topic:

NLS Load Options Overview

Novell License Certificate Format

The NLS license certificate format provides a prototype of the common certificate format. The common certificate format consists of two major categories of attributes.

The required attributes

The optional attributes

The required attributes make up the common portion of the license certificate and should adequately describe a licensed application's basic information.

The optional attributes provide licensing-system specific enhancements to the common portion of the license certificate. These enhancements are provided to add flexibility to the attributes supported by the common portion of the certificate.

The License Certificate Format documentation has these sections:

Required License Attributes

Optional License Attributes

Base License Type Descriptors

Unit Related Descriptors

Error Handling Descriptors

Default Descriptor Set

License Certificate Authentication and Security

Following a presentation of the structure for each attribute associated with the NLS license certificate are some examples of common licensing models and license certificate authentication and security.

Parent Topic:

Novell License Certificate Format

Required License Attributes

Each license certificate is composed of a number of required attributes or fields. They are:

Object ID Structure

Common Certificate Start

Publisher Name

Product Name

Version

Number of License Units

Secrets

Start Date

Expiration Date

Unique Identifier Field

Default Metered Field

Assignment Required Field

Default Period Update Field

Default Consumption Field

Licensing System-Specific Area

Parent Topic:

Novell License Certificate Format

Object ID Structure

The following syntax indicates the common object ID structure for each required field in the common license certificate.

```
  typedef LSLICTAGTTAG     {      LS_STR            nodeID[8];      LS_ULON
   format     LS_ULONG          milliSecs; // In low-high format     } LS
```

Parent Topic:

Novell License Certificate Format

Common Certificate Start

This field is the overall wrapper for the common certificate. It indicates that every byte contained in this area describes a license certificate.

Format

```
OID|Length|Lic Certificate
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521927, 100}
```

length

An LS_ULONG in low-high format, indicating the number of bytes contained in the Lic Certificate area. This is the maximum amount of bytes that should be parsed for common license information.

Lic Certificate

The series of bytes making up the common license certificate.

Parent Topic:

Novell License Certificate Format

Publisher Name

This field is a string that uniquely identifies the publisher (manufacturer) of the product. This string should be unique in the first 32 characters, but may be longer. A company name and/or trademark should be used.

Format

```
OID | length | strlen | string
```

OID

The Object ID is:

```
{{137, 65, 21,73}, LS_TCPIP_PROTOCOL, 785521928, 225}
```

length

An LS_ULONG in low-high format, indicating the number of bytes following this parameter for this field.

strlen

The number of bytes contained in the string parameter. This value is an LS_ULONG in low-high format.

string

A localized (potentially double-byte) null-terminated string indicating the publisher name for this license certificate.

Parent Topic:

Novell License Certificate Format

Product Name

This field is a string that uniquely identifies the product for the specified publisher. This string should be unique for this specific publisher in the first 32 characters, but may contain more. The product name should not have version specific information in it.

Format

```
OID | length | strlen | string
```

OID

The Object ID is :

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521929, 59}
```

length

An LS_ULONG in low-high format, indicating the number of bytes that follow this parameter for this field.

strlen

An LS_ULONG in low-high format, indicating the number of bytes in the parameter string.

string

A localized (potentially double byte) null terminated string indicating the product name for this license certificate.

Parent Topic:

Novell License Certificate Format

Version

This field is a string that uniquely identifies the version of the product for which this is a license. It must be unique for the specific product in the first 12 characters, but may contain more.

Format

```
OID | length | strlen | string
```

OID

The Object ID is:

```
{{137, 65, 21,73}, LS_TCPIP_PROTOCOL, 785521929, 983}
```

length

An LS_ULONG in low-high format, indicating the number of bytes that follow this parameter.

strlen

The number of bytes contained in the parameter string. This number is an LS_ULONG in low-high format.

string

A localized (potentially double-byte) null-terminated string indicating the version string for this license certificate.

Parent Topic:

Novell License Certificate Format

Number of License Units

This field specifies the number of license units that are to be installed into the licensing database. This field is given meaning by the LSRequest call made by an application. This number is an unsigned 32 bit integer. Any number of units may be specified up to, but not including 0xFFFFFFFF (LS_DEFAULT_UNITS).

IMPORTANT: Do not assume that one unit represents one instance of the application. If the value LS_DEFAULT_UNITS appears in the number of units field, the licensing system should install the appropriate number of units for one instance of the application into that particular licensing database.

Format

```
OID | length | numUnits
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521932, 316}
```

length

The size of this field. In this case, the value 4 represented in low-high format.

numUnits

An LS_ULONG in low-high format, indicating the total number of license units available from this certificate. Note that this value can be up to, but not including, 0xFFFFFFFF. 0xFFFFFFFF is re-interpreted as a different value by the licensing system.

Parent Topic:

Novell License Certificate Format

Secrets

This field holds the secrets for this application. The certificate provides space for four encrypted 32 bit unsigned integers, the minimum amount required for LSAPI v1.1 compliance. These numbers are somewhat sensitive in nature, so they need to be incorporated into the authentication information as well as encrypted. Note that this field, through the security scheme, also provides for the authentication of the information contained in the license certificate.

Format

```
OID | length | secretBuffer
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521931, 148}
```

length

An LS_ULONG indicating the length, in bytes, of the remaining parameters for this field. This value is in low-high format.

secretBuffer

A buffer that contains the encrypted data representing the secrets and the license information. See the section about security for details about the contents of this field.

Parent Topic:

Novell License Certificate Format

Start Date

This field holds the time (in seconds since January 1, 1970) at which the license is considered to be active and available. Note that a value of zero in this field indicates that the license has no specific start date and is always available for request.

Format

```
OID | length | date
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521933, 731}
```

length

An LS_ULONG indicating the length in bytes of the date field. It is currently fixed at 4 and should be represented in low-high format.

date

An LS_ULONG indicating the seconds since January 1, 1970. This value is in low-high format.

Parent Topic:

Novell License Certificate Format

Expiration Date

This field holds the time (in seconds since January 1, 1970) after which the license is considered to have expired and is no longer available for request. Note that a value of zero in this field indicates that the license has no specific end date and is always available for request.

Format

```
OID| length | date
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521934, 25}
```

length

An LS_ULONG indicating the length of the date field. This is fixed at 4 and should be represented in low-high format.

date

An LS_ULONG indicating the seconds since January 1, 1970. This value is in low-high format.

Parent Topic:

Novell License Certificate Format

Unique Identifier Field

This field, when used in combination with Publisher Name, Product Name, and Product Version, uniquely identifies the license. Since all these fields are utilized, the individual producer of the license may utilize any numbering scheme for uniquely identifying their licenses. Such options include serial number of the diskettes, time (down to the millisecond) the license was created, or license number created at that location. The goal of this field is to provide a means of manipulating individual license certificates for any given product installed in the licensing system.

Format

```
OID | Length | strlen | idBuffer
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521934, 894}
```

Length

An LS_ULONG in low-high format, indicating the length of the remaining parameters for this field.

strlen

The number of bytes contained in the parameter idBuffer. This value is an LS_ULONG in low-high format.

idBuffer

A buffer that contains a localized null terminated string holding the unique license identifier.

Parent Topic:

Novell License Certificate Format

Default Metered Field

This field indicates whether the license provided is considered a default metering license or not. If the value of this field is true, the licensing system should install the appropriate form of license for metering an application. If this flag is FALSE, the licensing system-specific Area describes the certificate's policy.

Format

```
OID | Length | Bool
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521936, 74}
```

Length

An LS_ULONG in low-high format, indicating the length of the boolean field. This value is fixed at 4.

Bool

An LS_ULONG in low-high format representing either TRUE (nonzero) or FALSE (zero).

Parent Topic:

Novell License Certificate Format

Assignment Required Field

This field specifies whether or not an assignment is required to use this license certificate. If this field is TRUE, then the certificate should not grant units until an assignment has been made to the certificate. If it is FALSE, the certificate should be immediately available but may accept assignments.

Format

```
OID | Length | Bool
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521937, 327}
```

Length

An LS_ULONG in low-high format, indicating the length of the boolean field. This value is fixed at 4.

Bool

An LS_ULONG in low-high format, representing either TRUE (nonzero) or FALSE (zero).

Parent Topic:

Novell License Certificate Format

Default Period Update Field

This field indicates the value that should be returned when an application queries for the default update period using LSQuery. This allows this policy attribute to change on a license by license basis. This value can be hard-coded into an application if desired.

Format

```
OID | Length | Update Period
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521946, 372}
```

Length

An LS_ULONG in low-high format, representing the length of the update period field. This value is fixed at 4.

Update Period

An LS_ULONG in low-high format, indicating the number of minutes recommended between updates. This value should not be less than 15 as defined by the LSAPI v1.1 document.

Parent Topic:

Novell License Certificate Format

Default Consumption Field

This field indicates the number of units that should be consumed from this certificate upon receiving an LSRequest with LS_DEFAULT_UNITS specified. This policy attribute can be placed in the license certificate, or it can be placed into the client code directly.

Format

```
OID | Length | Units
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521948, 193}
```

Length

An LS_ULONG in low-high format, indicating the size of the units field. This value is fixed at 4.

Units

An LS_ULONG in low-high format, indicating the number of units that should be consumed from the installed units when receiving a request for LS_DEFAULT_UNITS.

Parent Topic:

Novell License Certificate Format

Licensing System-Specific Area

This field allows licensing-system specific extensions to be added to a common license certificate. Each field in this area must follow a specific format. First, each field is prefixed with an Object ID that is unique to the licensing system vendor.

Immediately following this tag is the length of the information contained by this tag. This occupies the first 4 bytes (an LS_ULONG) immediately following the OID and be in standard network order.

In this way, specific features from multiple licensing systems may be accomadated in a single license certificate. When a licensing system is installing a common certificate, it ignores all tags in this area that it does not understand. It can't reject a certificate based on unrecognized information contained in this area. Further, no field in this area replaces an existing field in the common portion of the certificate. These fields enhance the existing fields and provide additional information.

Format

```
OID | Length | buffer
```

OID

The Object ID is:

```
{{137, 65, 21, 73}, LS_TCPIP_PROTOCOL, 785521945, 472}
```

Length

An LS_ULONG in low-high format, indicating the number of bytes contained in buffer.

buffer

A buffer containing licensing-system specific fields in a system in the following format

information

The information associated with the above specified Object ID. This information is licensing-system specific and interpretable only by the appropriate license service provider.

Parent Topic:

Novell License Certificate Format

Optional License Attributes

NLS-specific attributes provide enhanced functionality over the standard license certificate information. This enhanced functionality includes a large variety of policy descriptors. These policy descriptors enable the license to describe exactly how the application behaves in a variety of situations. Note that since all policy descriptors have their own tags, new descriptors can readily be added to a system, requiring only minor modifications to the license service provider. Also, the license service provider knows to reject any individual tag that it is not aware of, so backwards compatibility is guaranteed without the need for explicit versioning.

Descriptors, by their very nature, are optional. If present, they enhance the functionality of the license certificate. If no descriptors are present, a set of default descriptors is utilized. This default set is described after the policy descriptor descriptions. The following is a list of all of the current policy descriptors and their associated tags and information. Note that all tags (both common and NLS specific) are defined in "tags.h" provided in this SDK.

Parent Topic:

Novell License Certificate Format

Base License Type Descriptors

The following descriptors indicate base license types:

Consumptive Descriptor

Shareable Descriptor

Hardware Secured Descriptor

Public/Private Key Authentication Stamp (Digital Signature)

Parent Topic:

Novell License Certificate Format

Consumptive Descriptor

The presence of this descriptor indicates that when license units are requested from this certificate, the count of units is decremented, and those units will not be available again. This descriptor requires no arguments. Note that both Consumptive and Shareable can't be present at the same time, or an error occurs.

Format

```
TagNo | length | bool
```

TagNo

The value NLS_LIC_CONSUMABLE, in low-high format.

length

The size of the boolean field. Fixed at 4 and in low-high format.

bool

A boolean indicating if this descriptor is active or not. TRUE is any nonzero value; FALSE is zero.

Parent Topic:

Novell License Certificate Format

Shareable Descriptor

The presence of this descriptor indicates that when license units are requested from this certificate, the count of available units should be decremented, and when the units are released, the count should be incremented by the amount of units released (up to the amount originally installed).

Format

```
TagNo | length | bool
```

TagNo

The value NLS_LIC_SHAREABLE, in low-high format.

length

The size of the boolean field. Fixed at 4 and in low-high format.

bool

A boolean indicating if this descriptor is active or not. TRUE is any nonzero value; FALSE is zero.

Parent Topic:

Novell License Certificate Format

Hardware Secured Descriptor

The presence of this descriptor indicates that this license certificate is secured to a hardware device. When the license is requested, the license service provider determines if the appropriate hardware key is attached. If present, the units are potentially granted. If not, an error is returned to the requesting application.

This descriptor has two parameters. The first parameter is the developer ID of the hardware device as well as a challenge string to send to the device. The second parameter is the expected response string to the challenge query. The exact format of these two strings will be available at a later date.

Format

```
TagNo | length | qlen | query string | rlen | response string
```

TagNo

The value NLS_LIC_HW_SECURED, in low-high format.

length

The overall length of the entire field.

qlen

The number of bytes contained in the query string.

query string

The query string to send to the hardware device (exact format TBD).

rlen

The number of bytes contained in the expected response string.

response string

The expected return from the hardware device.

Parent Topic:

Novell License Certificate Format

Public/Private Key Authentication Stamp (Digital Signature)

The presence of this descriptor indicates that this license certificate's information is protected by an authentication stamp.

This authentication stamp consists of a hash of the common license information plus the license specific information for the NLS licensing system, encrypted with a private key. The application can check this authentication by performing the hash itself, then validating the contained authentication stamp with the matching public key.

Note that this hash only includes the common areas (excluding the secrets) and the NLS specific information. This is to avoid interference with other licensing system-specific areas.

Format

```
TagNo | length | hash len | encrypted hash
```

TagNo

The value NLS_LIC_PP_SECURED, in low-high format.

length

The number of bytes remaining in this field after this parameter, as an LS_ULONG in low-high format.

hash len

The number of bytes contained in the encrypted hash field as an LS_ULONG in low-high format.

encrypted hash

The private key encrypted hash of the information contained in the license certificate, for the common tags and the NLS-specific tags.

Parent Topic:

Novell License Certificate Format

Unit Related Descriptors

The following descriptors indicate information about the number of units relative to a license certificate:

Minimum Units to Consume

Maximum Units to Consume

Parent Topic:

Novell License Certificate Format

Minimum Units to Consume

This descriptor indicates the minimum number of units to consume, no matter what value is specified by the application. If the value specified by the application is greater than this value, this descriptor has no effect. It has one parameter, an LS_ULONG, which indicates the minimum number of units.

Format

```
TagNo | Length | numUnits
```

TagNo

The value NLS_LIC_MIN_UNITS_TO_CONSUME, in low-high format.

Length

The number of bytes contained in the rest of the parameters for this field, as an LS_ULONG in low-high format.

numUnits

An LS_ULONG in low-high format, indicating the minimum number of units to consume from this certificate.

Parent Topic:

Novell License Certificate Format

Maximum Units to Consume

This descriptor indicates the maximum number of units to consume, no matter what value is specified by the application. If the value specified by the application is less than this value, this descriptor has no effect. It has one parameter, an LS_ULONG, which indicates the maximum number of units.

Format

```
TagNo | Length | numUnits
```

TagNo

The value NLS_LIC_MAX_UNITS_TO_CONSUME, in low-high format.

Length

An LS_ULONG in low-high format, indicating the number of bytes remaining in this field after this parameter.

numUnits

An LS_ULONG in low-high format, indicating the maximum number of units to consume from this license certificate in any one request.

Parent Topic:

Novell License Certificate Format

Error Handling Descriptors

The following descriptors indicate how NLS responds to various errors:

Grant on LS_INSUFFICIENT_UNITS

Success on LS_LICENSE_TERMINATED

Grant on LS_LICENSE_UNAVAILABLE

Grant on LS_RESOURCES_UNAVAILABLE

Grant on LS_LICENSE_EXPIRED

Parent Topic:

Novell License Certificate Format

Grant on LS_INSUFFICIENT_UNITS

If this descriptor is present, when an LSRequest can't find enough installed units, it should return LS_SUCCESS anyway. All remaining units are consumed.

This descriptor has one parameter, an LS_ULONG. This parameter describes the number of additional license units to grant beyond the base amount. The licensing system no longer returns LS_SUCCESS if the number of consumed licenses exceeds the number of installed licenses by the specified number of units. If the value of the parameter is zero, no limitation occurs.

Note that the parameter value can represent a percentage factor by the license creation tool pre-computing the number of licenses the percentage represents.

Format

```
TagNo | Length | numUnits
```

TagNo

The value NLS_ERR_INSUF_UNITS, in low-high format.

Length

An LS_ULONG in low-high format, indicating the number of bytes remaining in this field after this parameter.

numUnits

The above described parameter as an LS_ULONG in low-high format.

Parent Topic:

Novell License Certificate Format

Success on LS_LICENSE_TERMINATED

This descriptor tells the licensing system to return success on an LSUpdate even if the license has been reclaimed for some reason.

For example, if an application fails to check back in with the licensing system, the licensing system may assume that the power has been shut off to the PC. It reclaims the license units and grants them to someone else. After this has occurred, suppose the application checks back in. This descriptor would enable the licensing system to return success.

This descriptor has one parameter, an LS_ULONG. This parameter describes the number of additional units that may be granted. The licensing system no longer returns LS_SUCCESS if the number of consumed licenses exceeds the number of installed licenses by this parameter. If a value of the parameter is zero, no limitation occurs.

Note that this value can represent a percentage factor by the license creation tool pre-computing the number of licenses the percentage represents.

Format

```
TagNo | Length | numUnits
```

TagNo

The value NLS_ERR_LIC_TERM, in low-high format.

Length

An LS_ULONG in low-high format, indicating the number of bytes remaining in this field after this parameter.

numUnits

The above described parameter as an LS_ULONG in low-high format.

Parent Topic:

Novell License Certificate Format

Grant on LS_LICENSE_UNAVAILABLE

If this descriptor is present, the licensing system returns LS_SUCCESS on an LSRequest or an LSUpdate if either of these calls would have generated the error code LS_LICENSE_UNAVAILABLE. Any available units are consumed/utilized.

This descriptor has one parameter, an LS_ULONG. This parameter describes the maximum number of additional units to grant. The licensing system no longer returns LS_SUCCESS if the number of consumed licenses exceeds the number of installed licenses by the specified value. If a value of the parameter is zero, no limitation occurs.

Note that this value can represet a percentage factor by the license creation tool pre-computing the number of licenses the percentage represents.

Format

```
TagNo | Length | numUnits
```

TagNo

The value NLS_ERR_LIC_UNAVAIL, in low-high format.

Length

An LS_ULONG in low-high format, indicating the number of bytes remaining in this field after this parameter.

numUnits

The above described parameter as an LS_ULONG in low-high format.

Parent Topic:

Novell License Certificate Format

Grant on LS_RESOURCES_UNAVAILABLE

If this descriptor is present, the licensing system returns LS_SUCCESS if any licensing call generates an out of memory/disk/etc. condition.

Note that the system can only detect this error if it occurs on the server. If the client runs out of memory before the request is sent to the server, the error is still returned.

This descriptor has one parameter, an LS_ULONG, operating as a boolean variable. A nonzero value indicates that this descriptor is active, a zero value indicates that this descriptor is not active (default).

Format

```
TagNo | Length | Bool
```

TagNo

The value NLS_ERR_RES_UNAVAIL, in low-high format.

Length

An LS_ULONG in low-high format, indicating the number of bytes remaining in this field following this parameter.

Bool

An LS_ULONG in low-high format. A nonzero value is TRUE; a zero value is FALSE.

Parent Topic:

Novell License Certificate Format

Grant on LS_LICENSE_EXPIRED

If this descriptor is present, the licensing system returns LS_SUCCESS if any applications requesting units encounters an LS_LICENSE_EXPIRED error. This only occurs when an installed license that would normally cause the request to be successful has expired.

This descriptor has one parameter, an LS_ULONG. This parameter indicates the maximum number of days for this grace period. If the value of this parameter is zero, there is no limit to the grace period.

Note that this value can represent a percentage by pre-computing the number of days the percentage would represent and placing it in this parameter.

Format

```
TagNo | Length | numDays
```

TagNo

The value NLS_ERR_LIC_EXPIRED, in low-high format.

Length

An LS_ULONG in low-high format, indicating the number of bytes remaining in this field after this parameter.

numDays

The above described parameter as an LS_ULONG in low-high format.

Parent Topic:

Novell License Certificate Format

Default Descriptor Set

The only NLS-specific descriptor that is active by default is the Shareable Descriptor. All other NLS descriptors are considered inactive unless otherwise present in the license certificate. The following examples show how to imitate common existing licensing models by combining the available descriptors.

Concurrent Licensing Model

Nodelocked Licensing Model (not currently available)

Personal Use Licensing Model

Site Licensing Model

One-Time Use

Parent Topic:

Novell License Certificate Format

Concurrent Licensing Model

Required License Descriptors

Default Units Descriptor <parameter value = 1> 1 unit represents one execution of the application.

Optional License Descriptors

Any additional descriptors that further define the application's policy. Note: the Consumptive Descriptor can't be present.

Parent Topic:

Novell License Certificate Format

Nodelocked Licensing Model (not currently available)

Required License Descriptors

NOTE: The license is located locally on the machine.

Default Metered Descriptor present and/or Shareable Descriptor present Assignment Required Descriptor <parameter value = 1>

NOTE: A machine's ID needs to be assigned to the license by the administrator.

Default Units Descriptor <parameter value = 1> 1 unit represents one execution of the application.

Optional License Descriptors

Any additional descriptors that further define the application's policy. Note: the Consumptive Descriptor can't be present.

Parent Topic:

Novell License Certificate Format

Personal Use Licensing Model

Required License Descriptors

Default Metered Descriptor present and/or Shareable Descriptor present Assignment Required Descriptor <parameter value = 1>

The administrator needs to assign a user to the license to activate it.

Default Units Descriptor <parameter value = 1> 1 unit represents one execution of the application.

Optional License Descriptors

Any additional descriptors that further define the application's policy. Note: the Consumptive Descriptor can't be present.

Parent Topic:

Novell License Certificate Format

Site Licensing Model

Required Descriptors

Default Metered Descriptor and/or Shareable Descriptor present Assignment Required Descriptor <parameter value = 1> An organization/tree name must be assigned to this license by the administrator.

Default Units Descriptor <parameter value = 1> 1 unit represents one execution of the application.

Grant on LS_INSUFFICIENT_UNITS <parameter value = 0> Makes the required license units fully available.

Grant on LS_LICENSE_TERMINATED <parameter value = 0> Users should always have access to the required units.

Optional Descriptors

Any additional license descriptors that further define the licensed application's policy. Note: the Consumptive Descriptor can't be present.

Parent Topic:

Novell License Certificate Format

One-Time Use

Required Descriptors

Consumptive Descriptor present Default Units Descriptor <parameter value = 1> 1 unit represents one execution of the application.

Optional Descriptors

Any additional license descriptors that further define the licensed application's policy. Note: the Shareable Descriptor can't be present.

Parent Topic:

Novell License Certificate Format

License Certificate Authentication and Security

There are two levels of authentication/security required for the implementation of secure license certificates. The first provides a reasonable implementation of secure license certificates. The first provides a reasonable level of security against license tampering via a common, easy to understand method. The second ensures a high degree of security against license tampering. This approach is more licensing-system specific.

LSAPI Authentication

NLS Specific Authentication

Parent Topic:

Novell License Certificate Format

LSAPI Authentication

The first level of authentication is contained in the common license certificate. The LSAPI v1.1 document requires four 32 bit secrets. These secrets need to be encrypted so they are difficult to decrypt, but decryptable by any licensing system. Further, the common license certificate needs to be protected from tampering,

The procedure for creating licenses requires that ISVs generate both a license certificate and an activation key. The license certificate/activation key pair provide authentication and encryption by the method in which they are utilized.

The activation key is created by taking the MD4 hash of a vendor determined string, the publisher name, and the time at which the license was created (in seconds). This hash becomes the activation key required by the license certificate.

Next in the certificate creation process is to encrypt the secrets with an encryption key. The encryption key is formed from the hash of the license information (minus the secrets and any licensing-system specific information) and the hash of the activation key. This hash then becomes the key to encrypt the secrets. Every time the licensing system needs to access the secrets, it must redetermine this encryption key.

If any information in the license certificate is changed, the keys can not be correctly decrypted because license-specific information is part of the encryption key. This provides security against license tampering. It also provides security against reading the secrets from the license.

Note that it is not too difficult to determine the secret given a valid activation key/certificate pair. Since the decryption process must be public, it is impossible to encrypt, then publicly decrypt these keys in a completely secure manner.

Creating an Activation Key

The following formula indicates the process described above for creating an activation key.

```
h('vendor determined string' + 'publisher name' + time) ---> activation
```

Encrypting/Decrypting Secrets

The following formulas indicate the process described above for encrypting and decrypting secrets.

```
h(license information + h(activation key)) ---> encryption key

f(encryption key, secrets) ---> encrypted secret/authentication buffer
```

NOTE: License information = common license certificate information only (not including the authentication field).

Parent Topic:

Novell License Certificate Format

NLS Specific Authentication

Because the common license certificate authentication scheme is imperfect, NLS provides a more secure authentication process. This process makes it very difficult to forge a certificate.

To ensure this high degree of confidence, public/private key pairs are utilized for authenticating the information contained in the license certificate.

In the licensing-system specific information area of the common license format, an additional tag is provided for NLS authentication information.

A message digest of the license information (minus the NLS authentication field and any other licensing system-specific information that is not NLS related) is computed. This message digest is then encrypted with the product vendor's private key. This result is stored in thePublic/Private Key Authentication Stamp (Digital Signature) descriptor in the licensing system-specific area of the license.

The application that uses this license can authenticate the validity of the license certificate by requesting the Public/Private Key Authentication Stamp (Digital Signature) using the **NLSGetCertificate** and **NLSCertificateTagToMessage** functions and then checking it using the corresponding public key.

Note that computing the private key from the public key, and therefore forging a license certificate, is very unlikely if a key size of 1024 bits is used.

Parent Topic:

Novell License Certificate Format

Novell Licensing: Functions

LSEnumProviders

Returns a unique string containing the name of any accessible licensing system

NetWare Server: 4.1

Platform: DOS, NLM, Windows* 3.1, Windows*95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE LSEnumProviders (
   LS_ULONG   index,
   LS_STR    *buffer);
```

Parameters

index

(IN) Specifies an index for all installed licensing systems (zero for the first licensing system, 1 for the second, and so on).*index* should be incremented by the caller for each successive call to **LSEnumProviders** until `LS_BAD_INDEX` is returned.

buffer

(OUT) Points to the buffer where the unique NULL-terminated string identifying the licensing system will be placed. This buffer must be at least 255 bytes.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

Remarks

An application can enumerate the installed licensing systems by calling **LSEnumProviders** successively. The string returned by **LSEnumProviders** is the same as that returned by calling **LSQuery** with information set to `LS_INFO_SYSTEM`.

LSFreeHandle

Releases all resources associated with the specified licensing context
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
void LSFreeHandle (
   LS_HANDLE   licenseHandle);
```

Parameters

licenseHandle

(IN) Specifies a handle, created by **LSRequest**, identifying a licensing context.

Return Values

None

Remarks

LSFreeHandle should be called after **LSRelease** returns or after an error is returned by **LSRequest**. **LSFreeHandle** releases all resources associated with the specified licensing context, including license units. Calls to **LSGetMessage** should be made before the handle is freed.

To successfully release license units, the user associated with the current connection must be security-equivalent to the original creator of the license handle (from **LSRequest**).

LSGetMessage

Gets a string describing the current error code

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE LSGetMessage (
   LS_HANDLE         licenseHandle,
   LS_STATUS_CODE    value,
   LS_HANDLE        *buffer,
   LS_ULONG         bufferSize);
```

Parameters

licenseHandle

(IN) Specifies the handle that identifies the license context. This argument must be a handle that was created with a successful or unsuccessful call to **LSRequest**.

value

(IN) Specifies any status code that can be returned by an LSAPI function (see NLS Status Codes).

buffer

(OUT) Points to a buffer where the localized error message string is returned.

bufferSize

(IN) Specifies the maximum size of information that licensing system returns to buffer.

Return Values

The resulting detailed status of the **LSGetMessage** function:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BAD_HANDLE`

licenseHandle did not describe a valid license system context.

Remarks

For a given error code, **LSGetMessage** returns a string describing the error. The string describes the error and a recommended action to take. If *value* is `LS_USE_LAST`, the last error associated with the supplied licensing handle is returned; otherwise, the supplied error code is used.

LSQuery

Gets information about a license certificate
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE LSQuery (
   LS_HANDLE    licenseHandle,
   LS_ULONG     information,
   LS_VOID     *infoBuffer,
   LS_ULONG     bufferSize,
   LS_ULONG    *actualBufferSize);
```

Parameters

licenseHandle

(IN) Specifies a handle identifying the license context. This must be a handle created by a successful call to **LSRequest**.

information

(IN) Identifies the information to be returned:

LS_INFO_NONE	Reserved
LS_INFO_SYSTEM	Returns the unique ID of the licensing system supplying the current license context. This ID is a NULL-terminated string equivalent to the value returned by **LSEnumProviders**.
LS_INFO_DATA	Returns the entire license certificate. The first LS_ULONG in the data buffer indicates the size in bytes of the data that follows.
LS_UPDATE_PERIOD	Returns the recommended interval in minutes at which **LSUpdate** should be called. This value is returned as an LS_ULONG in the data buffer. If a value of 0xFFFFFFFF is returned, no recommended update period exists for the associated system.

infoBuffer

(OUT) Points to a buffer where the resulting information is returned.

bufferSize

(IN) Specifies the maximum number of bytes for the licensing system to return in *infoBuffer.* This should be large enough to hold the expected data; otherwise, the status code `LS_BUFFER_TOO_SMALL` is returned, and only *bufferSize* bytes of data are returned.

actualBufferSize

(OUT) Points to the number of bytes of information actually put in *infoBuffer*. This value does not include any trailing NULL bytes.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_AUTHORIZATION_UNAVAILABLE`

Current user was not security-equivalent to the original creator of the licensing handle.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BAD HANDLE`

licenseHandle did not indicate a currently valid licensing handle.

Remarks

LSQuery gets information about license units obtained by calling **LSRequest**. For example, an application can determine the license type, time restrictions, and so on.

LSRelease

Releases licensing resources associated with the license context

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE LSRelease (
   LS_HANDLE   licenseHandle,
   LS_ULONG    totUnitsConsumed,
   LS_STR     *logComment);
```

Parameters

licenseHandle

(IN) Specifies a handle identifying the license context. *licenseHandle* must be a handle received from a successful call to **LSRequest**.

totUnitsConsumed

(IN) Specifies the total number of units consumed in this license handle context since **LSRequest** was initially called. The software publisher can specify this policy attribute within the application. A value of `LS_DEFAULT_UNITS` indicates that the licensing system should determine the appropriate value using its own licensing policy mechanisms.

logComment

(IN) Points to an optional string indicating a comment to be associated with the release and logged by NLS. The comment is logged even if an error is returned. To avoid logging the comment, specify `LS_NULL`.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_AUTHORIZATION_UNAVAILABLE`

Current user was not security-equivalent to the original creator of the licensing handle.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BAD HANDLE`

licenseHandle did not indicate a currently valid licensing handle.

Remarks

LSRelease releases licensing units associated with the license context identified by *licenseHandle*. The license handle context is not freed; to free the handle, call **LSFreeHandle**.

To successfully release license units, the user associated with the current connection must be security-equivalent to the original creator of the license handle (from **LSRequest**).

LSRequest

Requests licensing resources associated with a specific software product

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE LSRequest (
   LS_STR         *licenseSystem,
   LS_STR         *publisherName,
   LS_STR         *productName,
   LS_STR         *version,
   LS_ULONG        totUnitsReserved,
   LS_STR          logComment,
   LS_CHALLENGE   *challenge,
   LS_ULONG       *totUnitsGranted,
   LS_HANDLE      *licenseHandle);
```

Parameters

licenseSystem

(IN) Points to a string (from **LSEnumProviders**) that uniquely identifies a specific licensing system. `LS_ANY` indicates a match against all installed licensing systems.

publisherName

(IN) Points to the name of the publisher (manufacturer) of this product. This string must not be NULL and must be unique in the first 32 characters. A company name and trademark should be used to guarantee uniqueness.

productName

(IN) Points to the name of the product requesting license units. This string must not be NULL and must be unique in the first 32 characters in the *publisherName* domain.

version

(IN) Points to the version number of this product. This string must be unique in the first 12 characters in the *productName* domain. This string must not be NULL.

totUnitsReserved

(IN) Specifies the number of units required to run the application. The software publisher can specify this policy attribute within the application. `LS_DEFAULT_UNITS` lets the licensing system determine the number of units using information provided by the licensing system or the license certificate. NLS verifies that the requested number of units exist and can reserve those units, but no units are actually consumed at this time. The number of units available is returned in *totUnitsGranted*.

logComment

(IN) Specifies an optional string indicating a comment to be associated with the request and logged by NLS. The comment is logged even if an error is returned. To avoid logging the comment, specify `LS_NULL`.

challenge

(IN/OUT) On input, points to a challenge structure (`LS_NULL` if no challenge mechanism is desired). On output, points to the response to the challenge.

totUnitsGranted

(OUT) Points to the `LS_ULONG` where the total number of units granted is returned.

licenseHandle

(OUT) Points to the `LS_HANDLE` where a handle to the license context is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_AUTHORIZATION_UNAVAILABLE`

Current user was not security-equivalent to the original creator of the licensing handle.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_INSUFFICIENT_UNITS`

Licensing system could not find enough installed license units to satisfy the request at this time.

`LS_LICENSE_UNAVAILABLE`

Not enough license units are available to fulfill the request, but there are enough units installed to fulfill the request.

`LS_NETWORK_UNAVAILABLE`

The network is currently unavailable.

Remarks

If a valid license certificate is found and the challenge mechanism is used, the challenge response is computed and `LS_SUCCESS` is returned. At a minimum, the *publisherName*, *productName*, and *version* strings are used to identify matching license certificates.

For an **LSRequest** to be able to access a license certificate, the distinguished name (DN) associated with the client's connection must be security-equivalent to at least one of the assignments listed in the license certificate. If no assignments are listed, and the policy of the license certificate allows this, the license certificate can be accessed by anyone. NLS also determines user and machine information based on the NCP connection and the associated NDS information, and determines whether license units can be granted or not based on this information. Only license units that are accessible to the user are returned by this function.

The application should always call **LSFreeHandle** to release the memory associated with the specified handle. If license units are granted, the application must call **LSRelease** to release the granted units before **LSFreeHandle** is called.

LSUpdate

Verifies that the current handle is still valid
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE LSUpdate (
   LS_HANDLE       licenseHandle,
   LS_ULONG        totUnitsConsumed,
   LS_ULONG        totUnitsReserved,
   LS_STR         *logComment,
   LS_CHALLENGE   *challenge,
   LS_ULONG       *totUnitsGranted);
```

Parameters

licenseHandle

(IN) Specifies a handle identifying the license context. This argument must be a handle that was received from a successful call to **LSRequest**.

totUnits Consumed

(IN) Specifies the total number of units consumed so far in this handle context. The software publisher can specify this policy attribute within the application. `LS_DEFAULT_UNITS` indicates that the licensing system should determine the appropriate value, using its own licensing policy mechanisms. If an error is returned, the units are not consumed.

totUnitsReserved

(IN) Specifies the number of units to be reserved. If no additional units are required since the initial call to **LSRequest**, or since the last call to **LSUpdate**, then this parameter should be the current total as returned in *totUnitsGranted* from **LSRequest**. The total units reserved includes units consumed.

logComment

(IN) Points to an optional string indicating a comment to be associated with the request and logged by NLS. The comment is logged even if an error is returned. To avoid logging the comment, specify `LS_NULL`.

challenge

(IN/OUT) On input, points to a challenge structure (`LS_NULL` if no challenge mechanism is desired). On output, points to the response to the challenge.

totUnitsGranted

(OUT) Points to the total number of units granted since the initial license request was returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_AUTHORIZATION_UNAVAILABLE`

Current user was not security-equivalent to the original creator of the licensing handle.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BAD HANDLE`

licenseHandle did not indicate a currently valid licensing handle.

`LS_INSUFFICIENT_UNITS`

Licensing system could not find enough installed license units to satisfy the request at this time.

`LS_LICENSE_UNAVAILABLE`

Not enough license units are available to fulfill the request, but there are enough units installed to fulfill the request.

`LS_LICENSE_TERMINATED`

Re-verification of the existing units failed, so no additional units were requested. An update was not done within the specified update period, and the license units were issued to another user by NLS before this update was attempted.

`LS_LICENSE_EXPIRED`

Re-verification of the existing units failed due to an out-of-date license certificate. Additional units were not requested.

Remarks

The client application periodically issues this call to re-verify that the current license handle is still valid. **LSQuery** can be used to determine the proper update interval for the current licensing context. A guideline of once an hour might be appropriate, with a minimum interval of 15 minutes. By default, NLS utilizes the minimum time of once every 15 minutes. The software vendor can specify a different interval in the license certificate itself. Any update period specified in the license certificate overrides the default update period.

If the number of new units requested is greater than the number available, then the update request fails with an `LS_INSUFFICIENT_UNITS` error. On successful completion, **LSUpdate** returns *totUnitsGranted* to indicate the current running total of units granted. If *totUnitsConsumed* exceeds the number of units reserved, then the `LS_INSUFFICIENT_UNITS` error is returned, and the remaining units are consumed. A challenge response is not returned if an error occurs during the update process.

To successfully update license units, the user associated with the current connection must be security-equivalent to the original creator of the license handle (from **LSRequest**).

NLSAddAssignment

Restricts the use of a license certificate based on assignment type
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSAddAssignment (
   LS_STR          *licenseSystem,
   LS_STR          *publisher,
   LS_STR          *product,
   LS_STR          *version,
   LS_LICENSE_ID    licenseID,
   LS_ULONG         assignmentType,
   LS_VOID         *assignmentInfo,
   LS_ULONG        *assignmentInfoLen);
```

Parameters

licenseSystem

(IN) Points to the licensing system on which the specified product license certificate exists. It must be the specific name of a licensing system as returned by **LSEnumProviders**, or `LS_ANY`.

publisher

(IN) Points to the publisher name of the license certificate to which the assignment is added (not `LS_ANY`).

product

(IN) Points to the product name of the license certificate to which the assignment is added (not `LS_ANY`).

version

(IN) Points to the version string of the license certificate to which the assignment is added (not `LS_ANY`).

licenseID

(IN) Specifies the ID of the license certificate to which assignment is added.

assignmentType

(IN) Specifies the type of assignment to add in *assignmentInfo*. Current valid values and their associated *assignmentInfo* format are as follows:

`LS_USER_NAME` restricts based on a user or object name. NDS containers or group names can be specified. All elements that exist under a container or group then have access. *assignmentInfo* should contain the name to which access is being assigned.

`LS_MACHINE` restricts based on a machine address. This allows a license certificate to be assigned to one or more machines. *assignmentInfo* should contain the station address of the machine.

assignmentInfo

(IN) Points to the information required for performing the assignment based on the parameter *assignmentType*.

assignmentInfoLen

(IN) Points to the length of *assignmentInfo*, including the NULL terminator.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_AUTHORIZATION_UNAVAILABLE`

Either the license certificate specified could not be located, or the current user was not security-equivalent to the license certificate.

Remarks

NLSAddAssignment restricts the usage of a license certificate based on *assignmentType*. This allows license certificates to be reserved for a specific user, group, or machine. Additional restriction types will become available as their need is determined.

NLSAddAssignment applies restrictions to a specific license certificate, so all input must be fully specified. Also, this effect is additive. Additional calls to **NLSAddAssignment** cause the new restriction to be added in an “OR-like” fashion. To change the assignment to a different user, first remove the old restriction with**NLSRemoveAssignment**.

To add assignments, the user associated with the current connection must be security-equivalent to the owner attribute of the license certificate.

NLSAddProductInformation

Adds non-secure license certificate information to the licensing system database

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSAddProductInformation (
   LS_STR    *licenseSystem,
   LS_STR    *contextName,
   LS_STR    *publisherName,
   LS_STR    *productName,
   LS_STR    *version,
   LS_ULONG   licenseUnits,
   LS_ULONG   attributesLength,
   LS_VOID   *licenseAttributes);
```

Parameters

licenseSystem

(IN) Points to the name of the licensing system into which this product information should be installed. `LS_ANY` can be specified, indicating that this information is to be installed into the first licensing system that supports this procedure.

contextName

(IN) Points to the context where information will be added. `LS_ANY` can be specified, indicating that this information will be installed into the current user's context on the current connection.

publisherName

(IN) Points to the publisher name of the product being installed (cannot be NULL or `LS_ANY`).

productName

(IN) Points to the product name of the product being installed (cannot be NULL or `LS_ANY`).

version

(IN) Points to the version of the product being installed (cannot be NULL or `LS_ANY`).

licenseUnits

(IN) Specifies the number of license units to install for this product.

attributesLength

(IN) Specifies the length of the optional policy attribute section of this information to install. This should be set to zero if no attributes are specified.

licenseAttributes

(IN) Points to a buffer containing the attributes occurring contiguously. These override the corresponding default attributes in the licensing system.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

Remarks

NLSAddProductInformation allows a software asset management product (or an administrator) to create non-secure metering information in the licensing system databases. Specific policy attributes about the software's license agreement can be placed into the database along with the product information. This creates an "internal license" for the product, with no security information. These licenses might not be usable by a license-enabled application if the application requires the security information to execute. The installer of this information becomes the "owner" of this information.

NLS creates a unique license ID for license certificate information installed by **NLSAddProductInformation**.

NLSAvailable

Provides information about license units available for products in one or all licensing systems

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSAvailable (
   LS_STR          *licenseSystem,
   LS_STR          *contextName,
   LS_STR          *publisher,
   LS_STR          *product,
   LS_STR          *version,
   LS_ULONG         maxStrLen,
   LS_STR          *availablePublisher,
   LS_STR          *availableProduct,
   LS_STR          *availableVersion,
   LS_LICENSE_ID    availableLicID,
   LS_ULONG        *availableUnits,
   LS_ULONG        *index);
```

Parameters

licenseSystem

(IN) Points to the licensing systems to search for the specified products. It can either be the specific name of a licensing system, as returned by **LSEnumProviders**, or it can be `LS_ANY`. If `LS_ANY` is specified, information is returned from all licensing systems accessible from this client.

contextName

(IN) Points to the context to search for the specified products. It should be the distinguished name of the context, or `LS_ANY`. If `LS_ANY` is specified, the current user's context will be used. Information from the current context to the root is returned.

publisher

(IN) Points to the publisher name to filter by. Only transactions that have this publisher name are returned. `LS_ANY` indicates that any publisher name can appear in the returned entry.

product

(IN) Points to the product name to filter by. Only transactions that have this product name are returned. `LS_ANY` indicates that any product name can appear in the returned entry.

version

(IN) Points to the version string to filter by. Only transactions that have this version string are returned. `LS_ANY` indicates that any version string can appear in the returned entry.

maxStrLen

(IN) Specifies the maximum length (in bytes) of string output that can be placed into *availablePublisher*, *availableProduct*, and *availableVersion*. If any of these output strings is longer than *maxStrLen*, `LS_BUFFER_TOO_SMALL` is returned.

availablePublisher

(OUT) Points to the publisher name for which available units are returned. The product is completely described by a combination of *availablePublisher*, *availableProduct*, and *availableVersion*.

availableProduct

(OUT) Points to the product name for which the available units are returned.

availableVersion

(OUT) Points to the version for which the available units are returned.

availableLicID

(OUT) Returns the unique identifier/serial number of the license certificate from which the information was obtained, within the specific publisher, product, and version domain.

availableUnits

(OUT) Points to the number of available license units at the time the function was called. This does not necessarily indicate the number of installed instances of a product, because *availableUnits* might represent something else based on the application's licensing policy. Also, this takes into account the current connection status of the user calling **NLSAvailable**. If the certificate is not available, *availableUnits* is zero. When used in combination with **NLSInUse** and **NLSInstalled**, this condition can be easily detected.

index

(IN/OUT) Used to iterate through the entire list of publisher/product/version combinations that are currently available. *index* should be initialized to zero on the first call to the function, then opaquely passed back into the function on successive calls. To retrieve all information, continue this process until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_BAD_INDEX`

index does not point to a valid license certificate. This usually occurs when the end of the requested information has been completely returned.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

The *maxStrLen* parameter did not indicate a buffer large enough to hold one of the requested output parameters.

Remarks

NLSAvailable provides information about the number of license units available for products in one or all licensing systems, based on the filtering specified by *licenseSystem*, *publisher*, *product*, and *version*, on a certificate-by-certificate basis.

When **NLSAvailable** is first called, index should be initialized to zero. A single call to this function returns the availability of units for one certificate. To get information for other certificates, call **NLSAvailable** again with index reset to zero to obtain all the information relating to the newly specified parameters. When `LS_BAD_INDEX` is returned by **NLSAvailable**, all information relating to the specified filtering parameters has been returned.

If the license certificate has security restrictions associated with it (assignments, and so on), the number of units available to a non-security-equivalent object is always zero.

NLSCertificateTagToMessage

Presents license certificate information to the user in a more clear and organized manner

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSCertificateTagToMessage (
   LS_VOID    *licenseCertificate,
   LS_ULONG    licenseCertificateLength,
   LS_ULONG    maxStrLen,
   LS_STR     *messageString,
   LS_ULONG   *tagNumber);
```

Parameters

licenseCertificate

(IN) Points to a buffer that contains the data describing a valid license certificate. **NLSGetCertificate** can be used to retrieve this value.

licenseCertificateLength

(IN) Specifies the overall length of the license certificate (in bytes) pointed to by *licenseCertificate*.

maxStringLen

(IN) Specifies the maximum length of the string (in bytes) that can be placed into *messageString* on return from this function.

messageString

(OUT) Points to a buffer in which to place the localized message string on return. On output, it is a text message string describing the tag and its parameters.

tagNumber

(IN/OUT) Points to a tag number (in the order they appear in the certificate) for which the translation should be performed. To view all tags in a license certificate, this value should be initialized to zero on the first call to this function, and opaquely passed back until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_BAD_INDEX`

index does not point to a valid license certificate. This usually occurs when the end of the requested information has been completely returned.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

The *maxStrLen* parameter did not indicate a buffer large enough to hold one of the requested output parameters.

Remarks

The LSP performs this translation for the application; the application does not require any specific knowledge of the fields contained in the license certificate to perform this task.

NLSDeleteCertificate

Removes a license certificate record from the license database
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSDeleteCertificate (
   LS_STR          *licenseSystem,
   LS_STR          *publisherName,
   LS_STR          *productName,
   LS_STR          *version,
   LS_LICENSE_ID    licenseID);
```

Parameters

licenseSystem

(IN) Points to a licensing system that contains the license certificate to be deleted. Specify a specific licensing system name as received from **LSEnumProviders**, or specify `LS_ANY` to initiate a search for the first matching certificate.

publisherName

(IN) Points to the publisher name of the license certificate to delete from the license certificate database (cannot be `NULL` or `LS_ANY`).

productName

(IN) Points to the product name of the license certificate to delete from the license certificate database (cannot be `NULL` or `LS_ANY`).

version

(IN) Points to the version of the license certificate to delete from the license certificate database (cannot be `NULL` or `LS_ANY`).

licenseID

(IN) Specifies the unique identifier (serial number) of the license certificate to delete from the license certificate database.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_AUTHORIZATION_UNAVAILABLE`

Specified license certificate could not be located, or user does not have privileges to delete this license certificate.

Remarks

NLSDeleteCertificate removes the license certificate record from the license database where it is installed, or marks it as unusable. This license certificate is then no longer visible to any other licensing functions.

The user associated with the current connection must be security-equivalent to the owner attribute of the license certificate.

NLSGetCertificate

Retrieves the data that makes up a license certificate

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSGetCertificate (
   LS_STR          *licenseSystem,
   LS_STR          *publisherName,
   LS_STR          *productName,
   LS_STR          *version,
   LS_LICENSE_ID    licenseID,
   LS_ULONG         maxBufferLen,
   LS_VOID         *licenseCertificate,
   LS_ULONG        *actualBufferLen);
```

Parameters

licenseSystem

(IN) Points to the licensing system in which the license certificate to retrieve is located. Specify a specific licensing system name as received from **LSEnumProviders**, or specify LS_ANY to initiate a search for the first matching certificate.

publisherName

(IN) Points to the publisher name of the license certificate to get (cannot be NULL or LS_ANY).

productName

(IN) Points to the product name of the license certificate to get (cannot be NULL or LS_ANY).

version

(IN) Points to the version string of the license certificate to get (cannot be NULL or LS_ANY).

licenseID

(IN) Specifies the license certificate ID or serial number of the license certificate to get.

maxBufferLen

(IN) Specifies the maximum amount of data (in bytes) that can be placed into the buffer pointed to by *licenseCertificate*.

licenseCertificate

(OUT) Points to a buffer in which the license certificate is to be placed.

This buffer is filled only if the return is `LS_SUCCESS`.

actualBufferLen

(OUT) Points to the actual number of bytes the license certificate inhabits in the buffer pointed to by *licenseCertificate*.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

The *maxStrLen* parameter did not indicate a buffer large enough to hold one of the requested output parameters.

`LS_AUTHORIZATION_UNAVAILABLE`

The specified license certificate could not be located, or you do not have privileges to delete this license certificate.

Remarks

NLSGetCertificate lets an application retrieve the actual data comprising a license certificate. This data is in the standard license certificate format. The application needs to parse this data to obtain information about the license certificate or call **NLSCertificateTagToMessage** to translate the information, on a tag-by-tag basis, to localized message strings for display to the user.

NLSGetEntry

Views all the individual components of a transaction

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSGetEntry (
   LS_HANDLE       *transactionHandle,
   LS_TRANS_ENTRY  *transactionEntry,
   LS_ULONG        *index);
```

Parameters

transactionHandle

(IN) Points to the handle (returned by **NLSGetTransaction**) that points to a set of entries.

transactionEntry

(OUT) Points to an individual entry in the transaction. The entry is a structure of the following format:

```
struct LS_TRANS_ENTRY_TAG
{
   LS_STR          *actionName;
   LS_ULONG         time;
   LS_STATUS_CODE   actionResult;
   LS_ULONG         numUnits;
   LS_STR          *userLoggedMessage;
} LS_TRANS_ENTRY;
```

IMPORTANT: *actionName* and *userLoggedMessage* are allocated by the NLSAPI library. These values should be freed when the entry is no longer needed.

index

(IN/OUT) Used to iterate through the entire list of transactions. This should be initialized to zero on the first call to the function, then opaquely passed back into the function on successive calls. To retrieve all information, continue this process until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_HANDLE`

Specified handle does not describe an existing transaction handle.

`LS_BAD INDEX`

There are no more entries after the specified index.

Remarks

NLSGetEntry views all the individual components of a transaction. This lets you determine how long license units were held, and it provides a detailed history of any errors that occurred in the process of obtaining and holding license units. Most transaction entries are generated by applications calling LSAPI functions; however, system errors and license certificate installations also generate transactions.

The user associated with the current connection must be security-equivalent to the creator of the transaction handle.

NLSGetTransaction

Retrieves a single transaction record

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSGetTransaction (
   LS_STR     *licenseSystem,
   LS_STR     *publisher,
   LS_STR     *product,
   LS_STR     *version,
   LS_STR     *userName
   LS_ULONG    maxStrLen,
   LS_HANDLE  *transactionHandle,
   LS_STR     *transactionPublisher,
   LS_STR     *transactionProduct,
   LS_STR     *transactionVersion,
   LS_STR     *transactionUserName,
   LS_ULONG   *index);
```

Parameters

licenseSystem

(IN) Points to the licensing system to receive transaction entries from. Specify a specific licensing system name as received from **LSEnumProviders**, or specify `LS_ANY` to initiate a search for the first matching certificate.

publisher

(IN) Points to the publisher name to filter by. Only transactions that have this publisher name are returned. `LS_ANY` indicates that transactions will not be filtered by publisher.

product

(IN) Points to the product name to filter by. Only transactions that have this product name are returned. `LS_ANY` indicates that transactions will not be filtered by product.

version

(IN) Points to the version string to filter by. Only transactions that have this version string are returned. `LS_ANY` indicates that transactions will not be filtered by version.

userName

(IN) Points to the user/object name to filter by. Only transactions that have this user/object name are returned. `LS_ANY` indicates that transactions will not be filtered by user/object.

maxStrLen

(IN) Specifies the maximum length (in bytes) of string output that can be placed into *transactionPublisher*, *transactionProduct*, and *transactionVersion*. If any of these output strings is longer than *maxStrLen*, `LS_BUFFER_TOO_SMALL` is returned.

transactionHandle

(OUT) Points to the handle for the transaction. The individual entries for the overall transaction can then be obtained by calling **NLSGetEntry** with this handle. The handle is only valid when `LS_SUCCESS` is returned.

transactionPublisher

(OUT) Points to the name of the publisher for this transaction. This name applies to all entries in the returned transaction.

transactionProduct

(OUT) Points to the name of the product for this transaction. This name applies to all entries in the returned transaction.

transactionVersion

(OUT) Points to the version string for this transaction. This version string applies to all entries in the returned transaction.

transactionUserName

(OUT) Points to the user/object name that was responsible for initiating this transaction. This name applies to all entries in the returned transaction.

index

(IN/OUT) Used to iterate through the entire list of transactions.*index* should be initialized to zero on the first call to the function, then opaquely passed back into the function on successive calls. To retrieve all information, continue this process until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

The *maxStrLen* parameter did not indicate a buffer large enough to hold one of the requested output parameters.

`LS_AUTHORIZATION_UNAVAILABLE`

Specified license certificate could not be located, or you do not have privileges to delete this license certificate.

`LS_BAD INDEX`

There are no more entries after the specified index.

Remarks

NLSGetTransaction retrieves a single transaction record based on the filtering *licenseSystem*, *publisher*, *product*, *version*, and *userName*. This single transaction describes all of the individual calls a single licensed application made, including errors generated, from the initial call to **LSRequest** through the call to **LSFreeHandle**. It also includes license certificate installation and deletion.

The transaction handle is a one-for-one mapping with the license handle. The individual entries can be accessed with **NLSGetEntry**.

NLSInstallCertificate

Installs a license certificate into the specified licensing system

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSInstallCertificate (
   LS_STR     *licenseSystem,
   LS_STR     *contextName,
   LS_ULONG    licenseLength,
   LS_VOID    *licenseCertificate,
   LS_ULONG    passwordLength,
   LS_VOID    *activationPassword);
```

Parameters

licenseSystem

(IN) Points to the licensing system in which to install the license certificate. This should be either a value retrieved by **LSEnumProviders** or `LS_ANY`. If `LS_ANY` is specified, the licensing systems are tried in turn until the license certificate is successfully installed.

contextName

(IN) Points to the context where the license certificate will be added. `LS_ANY` can be specified, indicating that this license certificate will be installed into the current user's context on the current connection.

licenseLength

(IN) Specifies the total length of the license certificate to install, in bytes.

licenseCertificate

(IN) Points to a block of data representing a license certificate in the standard license certificate format.

passwordLength

(IN) Specifies the length of the activation password for the provided license certificate.

activationPassword

(IN) Points to the password required for activating the license certificate information provided in *licenseCertificate*.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_AUTHORIZATION_UNAVAILABLE`

Specified license certificate could not be located, or you do not have privileges to delete this license certificate.

Remarks

NLSInstallCertificate installs a license certificate into the specified licensing system and into the database specified by *contextName*. The license certificate is “activated” with the provided activation password. This password initializes the authentication information that can be associated with a license certificate.

The licensing system initially assigns ownership of the license certificate to the user who invoked this procedure. To change ownership, call **NLSTransferOwnership**.

NLSInstalled

Provides information about license units that are installed for products in one or all licensing systems

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSInstalled (
   LS_STR         *licenseSystem,
   LS_STR         *contextName,
   LS_STR         *publisher,
   LS_STR         *product,
   LS_STR         *version,
   LS_ULONG        maxStrLen,
   LS_STR         *installedPublisher,
   LS_STR         *installedProduct,
   LS_STR         *installedVersion,
   LS_LICENSE_ID   installedLicenseID,
   LS_ULONG       *numInstalledUnits,
   LS_ULONG       *index);
```

Parameters

licenseSystem

(IN) Points to the licensing systems to search for the specified products. It can either be the specific name of a licensing system, as returned by **LSEnumProviders**, or it can be LS_ANY. If LS_ANY is specified, information is returned from all licensing systems accessible from this client.

contextName

(IN) Points to the context to search for the specified products. It should be the distinguished name of the context, or LS_ANY. If LS_ANY is specified, the current user's context will be used. Information from the current context to the root is returned.

publisher

(IN) Points to the publisher name to filter by. Only transactions that have this publisher name are returned. LS_ANY indicates that any publisher name can appear in the returned entry.

product

(IN) Points to the product name to filter by. Only transactions that have this product name are returned. LS_ANY indicates that any product name can appear in the returned entry.

version

(IN) Points to the version string to filter by. Only transactions that have this version string are returned. `LS_ANY` indicates that any version string can appear in the returned entry.

maxStrLen

(IN) Specifies the maximum length (in bytes) of string output that can be placed into *availablePublisher*, *availableProduct*, and *availableVersion*. If any of these output strings is longer than *maxStrLen*, `LS_BUFFER_TOO_SMALL` is returned.

installedPublisher

(OUT) Points to the name of an installed publisher, based on current value of *index*.

installedProduct

(OUT) Points to the name of an installed product, based on current value of *index*.

installedVersion

(OUT) Points to the current installed version, based on the current value of *index*.

installedLicID

(OUT) Returns the unique identifier/serial number (within specified publisher, product, version domain) of the license certificate from which this information was retrieved.

numInstalledUnits

(OUT) Points to the number of available license units at the time the function was called. This does not necessarily indicate the number of installed instances of a product, because *availableUnits* might represent something else based on the application's licensing policy. Also, this takes into account the current connection status of the user calling **NLSAvailable**. If the certificate is not available, *availableUnits* is zero. When used in combination with **NLSInUse** and **NLSInstalled**, this condition can be easily detected.

index

(IN/OUT) Used to iterate through the entire list of publisher/product/version combinations that are currently available. *index* should be initialized to zero on the first call to the function, then opaquely passed back into the function on successive calls. To retrieve all information, continue this process until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_BAD INDEX`

Index does not point to a valid license certificate. This usually occurs when the end of the requested information has been completely returned.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

maxStrLen did not indicate a buffer large enough to hold one of the requested output parameters.

Remarks

NLSInstalled determines what products are available from a client. Filtering can be performed to get information for a specific product. If the filtering is changed, the index parameter should be reset to zero to get all information relating to the newly specified parameters. When `LS_BAD_INDEX` is returned by **NLSInstalled**, all information relating to the specified filtering parameters has been returned.

NLSInUse

Provides information about license units that are in use for products in one or all licensing systems

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSInUse (
   LS_STR          *licenseSystem,
   LS_STR          *contextName,
   LS_STR          *publisher,
   LS_STR          *product,
   LS_STR          *version,
   LS_ULONG         maxStrLen,
   LS_STR          *publisherInUse,
   LS_STR          *productInUse,
   LS_STR          *versionInUse,
   LS_LICENSE_ID    licIDInUse,
   LS_ULONG        *numUnitsInUse,
   LS_ULONG        *index);
```

Parameters

licenseSystem

(IN) Points to the licensing system to use to search for the specified products. It can either be the specific name of a licensing system, as returned by **LSEnumProviders**, or it can be `LS_ANY`. If `LS_ANY` is specified, information from all installed licensing systems on this client is returned.

contextName

(IN) Points to the context to search for the specified products. It should be the distinguished name of the context, or `LS_ANY`. If `LS_ANY` is specified, the current user's context will be used. Information from the current context to the root is returned.

publisher

(IN) Points to the publisher name to filter by. Only transactions that have this publisher name are returned. `LS_ANY` indicates that any publisher name can appear in the returned entry.

product

(IN) Points to the product name to filter by. Only transactions that have this product name are returned. `LS_ANY` indicates that any product name can appear in the returned entry.

version

(IN) Points to the version string to filter by. Only transactions that have this version string are returned. `LS_ANY` indicates that any version string can appear in the returned entry.

maxStrLen

(IN) Specifies the maximum length (in bytes) of string output that can be placed into *availablePublisher*, *availableProduct*, and *availableVersion*. If any of these output strings is longer than *maxStrLen*, `LS_BUFFER_TOO_SMALL` is returned.

publisherInUse

(OUT) Points to the name of the publisher for which usage information is being returned. Note that to uniquely identify a product, the *publisherInUse*, *productInUse*, and *versionInUse* must all be utilized.

productInUse

(OUT) Points to the name of the product for which usage information is being returned.

versionInUse

(OUT) Points to the version string for which usage information is being returned.

licIDInUse

(OUT) Returns the unique identifier/serial number (within the specified publisher, product, version domain) of the license certificate from which this information was retrieved.

numUnitsInUse

(OUT) Points to the current number of units that are in use for the returned *publisherInUse*, *productInUse*, and *versionInUse*. If `LS_ANY` was specified for *licenseSystem*, this number potentially crosses licensing systems. This value is not necessarily the number of installed instances of a product, because units might represent something else based on the application's licensing policy. Also, this takes into account the current connection status of the user calling **NLSInstalled**. If the certificate is not available, *numInstalledUnits* is zero. When used in combination with **NLSInUse** and **NLSAvailable**, this condition can be easily detected.

index

(IN/OUT) Used to iterate through the entire list of publisher/product/version combinations that are currently available. *index* should be initialized to zero on the first call to the function, then opaquely passed back into the function on successive calls. To retrieve all information, continue this process until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_BAD INDEX`

Index does not point to a valid license certificate. This usually occurs when the end of the requested information has been completely returned.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

The *maxStrLen* parameter did not indicate a buffer large enough to hold one of the requested output parameters.

Remarks

NLSInUse provides information about the number of license units available for products in one or all licensing systems, based on the filtering specified by *licenseSystem*, *publisher*, *product*, and *version*, on a certificate-by-certificate basis.

When **NLSInUse** is first called, *index* should be initialized to zero. A single call to this function returns the availability of units for one certificate. To get information for other certificates, call **NLSAvailable** again with *index* reset to zero to obtain all the information relating to the newly specified parameters. When `LS_BAD_INDEX` is returned by **NLSAvailable**, all information relating to the specified filtering parameters has been returned.

NLSMoveCertificate

Relocates the database record containing the specified license certificate
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSMoveCertificate (
   LS_STR         *licenseSystem,
   LS_STR         *publisherName,
   LS_STR         *productName,
   LS_STR         *version,
   LS_LICENSE_ID   licenseID,
   LS_STR         *newContextName);
```

Parameters

licenseSystem

(IN) Points to the licensing system in which the license certificate to move is located. Specify a specific licensing system name as received from **LSEnumProviders**, or specify `LS_ANY` to initiate a search for the first matching certificate.

publisher

(IN) Points to the publisher name of the license certificate to move to another server. This value is required and cannot be `LS_ANY`.

product

(IN) Points to the product name of the license certificate to move to another server. This value is required and cannot be `LS_ANY`.

version

(IN) Points to the version of the license certificate to move to another server. This value is required and cannot be `LS_ANY`.

licenseID

(IN) Specifies the unique identifier (serial number) of the license certificate to move to another server.

newContextName

(IN) Points to the context to which the license certificate will be moved. `LS_ANY` can be specified, indicating that this license certificate will be moved to the current user's context on the current connection.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_AUTHORIZATION_UNAVAILABLE`

Specified license certificate could not be located, or you do not have privileges to delete this license certificate.

Remarks

NLSMoveCertificate relocates the license certificate object to the specified context name. If this procedure cannot be completed successfully, the license certificate record remains in the original location.

The user associated with the current connection must be security-equivalent to the owner attribute of the license certificate and must have rights to place a license certificate on the new server specified.

NLSRemoveAssignment

Removes an assignment from a license certificate

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSRemoveAssignment (
   LS_STR          *licenseSystem,
   LS_STR          *publisher,
   LS_STR          *product,
   LS_STR          *version,
   LS_LICENSE_ID    licenseID,
   LS_ULONG         assignmentType,
   LS_VOID         *assignmentInfo,
   LS_ULONG         assignmentInfoLen);
```

Parameters

licenseSystem

(IN) Points to the licensing systems on which the specified product license certificate exists. It must be the specific name of a licensing system as returned by **LSEnumProviders**.

publisher

(IN) Points to the publisher name of the license certificate from which the assignment is removed.

product

(IN) Points to the product name of the license certificate from which the assignment is removed.

version

(IN) Points to the version string of the license certificate from which the assignment is removed.

licenseID

(IN) Specifies the unique identifier (serial number) of the license certificate from which the assignment is removed.

assignmentType

(IN) Specifies the type of assignment to remove. The information placed in *assignmentInfo* depends on *assignmentType*.

assignmentInfo

(IN) Points to the exact information for the assignment to remove.

assignmentInfoLen

(IN) Specifies the length of *assignmentInfo* including the NULL terminator.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_AUTHORIZATION_UNAVAILABLE`

Specified license certificate could not be located, or you do not have privileges to delete this license certificate.

Remarks

NLSRemoveAssignment removes an assignment from a license certificate that was added with **NLSAddAssignment**. All information about the assignment you wish to remove must be specified. This information can be obtained from the assignment list obtainable by calling **NLSGetLicense**. This function removes only one assignment from the assignment list. To remove all assignments, call this function multiple times on all the installed assignments.

The user associated with the current connection must be security-equivalent to the owner attribute of the license certificate.

NLSTransferOwnership

Transfers the ownership of a license certificate
NetWare Server: 4.1
Platform: DOS, NLM, Windows 3.1, Windows95
Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSTransferOwnership (
   LS_STR          *licenseSystem,
   LS_STR          *publisher,
   LS_STR          *product,
   LS_STR          *version,
   LS_LICENSE_ID    licenseID,
   LS_STR          *newOwner);
```

Parameters

licenseSystem

(IN) Points to the licensing systems on which the specified product license certificate exists. It must be the specific name of a licensing system as returned by **LSEnumProviders**.

publisher

(IN) Points to the publisher name of the license certificate from which the assignment is removed.

product

(IN) Points to the product name of the license certificate from which the assignment is removed.

version

(IN) Points to the version string of the license certificate from which the assignment is removed.

licenseID

(IN) Specifies the unique identifier (serial number) of the license certificate from which to transfer ownership.

newOwner

(IN) Points to the DN of the new owner of the license certificate.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_RESOURCES_UNAVAILABLE`

Insufficient resources (such as memory) available to complete request.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_AUTHORIZATION_UNAVAILABLE`

The specified license certificate could not be located, or you do not have privileges to delete this license certificate.

Remarks

NLSTransferOwnership transfers the ownership of the license certificate from its current value to the DN passed in through the *newOwner* parameter. Only users with a security equivalency to this new DN are able to add, modify, or remove assignments and delete the license certificate.

The user associated with the current connection must be security-equivalent to the owner attribute of the license certificate.

NLSUsers

Provides information about who or what is using licensing units for a product, in the specified licensing systems

NetWare Server: 4.1

Platform: DOS, NLM, Windows 3.1, Windows95

Service: Novell Licensing System (NLS)

Syntax

```
LS_STATUS_CODE NLSUsers (
   LS_STR          *licenseSystem,
   LS_STR          *contextName,
   LS_STR          *publisher,
   LS_STR          *product,
   LS_STR          *version,
   LS_ULONG         maxStrLen,
   LS_STR          *publisherInUse,
   LS_STR          *productInUse,
   LS_STR          *versionInUse,
   LS_LICENSE_ID    licIDInUse,
   LS_ULONG        *unitsInUse,
   LS_STR          *user,
   LS_ULONG         *index);
```

Parameters

licenseSystem

(IN) Points to the licensing systems in which to search for the specified products. It can either be the specific name of a licensing system, as returned by **LSEnumProviders**, or it can be `LS_ANY`. If `LS_ANY` is specified, information from all installed licensing systems on this client is returned.

contextName

(IN) Points to the context to search for the specified products. It should be the distinguished name of the context, or `LS_ANY`. If `LS_ANY` is specified, the current user's context will be used. Information from the current context to the root is returned.

publisher

(IN) Points to the publisher name to filter by. Only transactions that have this publisher name are returned. `LS_ANY` indicates that any publisher name can appear in the returned entry.

product

(IN) Points to the product name to filter by. Only transactions that have this product name are returned. `LS_ANY` indicates that any product name can appear in the returned entry.

version

(IN) Points to the version string to filter by. Only transactions that have this version string are returned. `LS_ANY` indicates that any version string can appear in the returned entry.

maxStrLen

(IN) Specifies the maximum length (in bytes) of string output that can be placed into *availablePublisher*, *availableProduct*, and *availableVersion*. If any of these output strings is longer than *maxStrLen*, `LS_BUFFER_TOO_SMALL` is returned.

publisherInUse

(OUT) Points to the name of the publisher for which usage information is being returned. Note that to uniquely identify a product, the *publisherInUse*, *productInUse*, and *versionInUse* must all be utilized.

productInUse

(OUT) Points to the name of the product for which usage information is being returned.

versionInUse

(OUT) Points to the version string for which usage information is being returned.

licIDInUse

(OUT) Returns the unique identifier/serial number (within the specified publisher, product, version domain) of the license certificate from which this information was retrieved.

unitsInUse

(OUT) Points to the current number of units that are in use for the returned *publisherInUse*, *productInUse*, and *versionInUse*. If `LS_ANY` was specified for *licenseSystem*, this number potentially crosses licensing systems. This value is not necessarily the number of installed instances of a product, because units might represent something else based on the application's licensing policy. Also, this takes into account the current connection status of the user calling **NLSInstalled**. If the certificate is not available, *numInstalledUnits* is zero. When used in combination with **NLSInUse** and **NLSAvailable**, this condition can be easily detected.

user

(OUT) Points to the DN of the user who is using license units of the returned product.

index

(IN/OUT) Used to iterate through the entire list of publisher/product/version combinations that are currently in use. index should be initialized to zero on the first call to the function, then opaquely passed back into the function on successive calls. To retrieve all information, continue this process until `LS_BAD_INDEX` is returned.

Return Values

Returns a detailed error code that can be directly processed by the caller, or that can be converted by **LSGetMessage** to one of these message strings:

`LS_SUCCESS`

The requested functionality completed successfully.

`LS_SYSTEM_UNAVAILABLE`

DOS TSR or Windows DLL is not properly configured or available, or client has no licensing system to communicate with.

`LS_BAD INDEX`

Index does not point to a valid license certificate. This usually occurs when the end of the requested information has been completely returned.

`LS_BAD_ARG`

One or more parameters are invalid.

`LS_BUFFER_TOO_SMALL`

The *maxStrLen* parameter did not indicate a buffer large enough to hold one of the requested output parameters.

Remarks

NLSUsers provides information about who or what is using licensing resources for a product and how many license units are being used by this person/object for one or all licensing systems. The returned information is based on the filtering specified by *licenseSystem*, *publisher*, *product*, and *version*, on a certificate-by-certificate basis.

On the first call to this function, *index* should be initialized to zero. A single call to this function returns a single user/object for one product. Subsequent calls either return another user for the same product or the next product if there are no more users/objects. To retrieve this additional information, make another call opaquely passing back *index*. If the filtering is changed, the *index* parameter should be reset to zero to get all the information relating to the newly specified parameters. When `LS_BAD_INDEX` is returned, all information relating to the specified parameters has been returned.

NWCalls Management

NWCalls Management: Functions

NWCallsInit

Initializes double-byte tables and low-level interface functions
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: NWCalls Management

Syntax

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

NWCCODE N_API NWCallsInit (
   nptr   in,
   nptr   out);
```

Pascal Syntax

```
#include <nwmisc.inc>

Function NWCallsInit
  (inPtr : nptr;
   outPtr : nptr
) : NWCCODE;
```

Parameters

NULL should be passed for both the *in* and *out* parameter pointers.

Return Values

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

Swapped word	SUCCESSFUL
1	Unable to Obtain Double Byte Information
0x88F	SHELL_FAILURE/REM_FAILURE

Remarks

NWCallsInit operates without double byte support.

For all client platforms, **NWCallsInit** initializes the unicode tables to the native country and code pages defined by the operating system.

For OS/2 and Windows 3.1, **NWCallsInit** is called by the startup code. The application should still call **NWCallsInit**, however, to check if the initialization succeeded.

For DOS, **NWCallsInit** must be called before any other NetWare® Cross-Platform API function.

NCP Calls

None

NWCallsTerm

Terminates the NWCalls library and performs any necessary clean up
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: NWCalls Management

Syntax

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

N_EXTERN_LIBRARY (NWCCODE)NWCallsTerm (
   nptr   reserved);
```

Pascal Syntax

```
#include <nwmisc.inc>

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

Parameters

reserved

(IN) Is reserved (pass NULL).

Return Values

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

0x0000	SUCCESSFUL
0x89FF	Failure

NCP Calls

None

NWGetNWCallsVersion

Returns the version number of the NWCalls library running on the calling entity

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

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

Service: NWCalls Management

Syntax

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

void N_API NWGetNWCallsVersion (
   pnuint8   majorVer,
   pnuint8   minorVer,
   pnuint8   revLevel,
   pnuint8   betaLevel);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWGetNWCallsVersion
  (majorVer : pnuint8;
   minorVer : pnuint8;
   revLevel : pnuint8;
   betaLevel : pnuint8
);
```

Parameters

majorVer

(OUT) Points to the major version number of the requester or shell.

minorVer

(OUT) Points to the minor version number of the requester or shell.

revLevel

(OUT) Points to the revision number of the requester or shell.

betaLevel

(OUT) Points to the beta revision number of the requester or shell.

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL

NCP Calls

None

NWSNMP

NWSNMP: Guides

NWSNMP: Task Guide

NWSNMP: Concept Guide

Parent Topic:

NWSNMP: Guides

NWSNMP Introduction

Simple Network Management Protocol (SNMP) is the protocol that offers network management services within the Internet protocol suite. SNMP management functions are actually independent of the Internet protocols, making it popular for management in many environments. Most leading network products support SNMP, allowing vendor-independent management of devices and applications.

The NWSNMP allows NWLoadable Module™ (NLM™) developers to accelerate development of NLM applications that are manageable by SNMP. Once you make information about your NLM accessible to NWSNMP, any SNMP remote client can be used to manage your NLM. Using the NWSNMP, you can make your NLM applications SNMP-manageable even if you have minimal knowledge about how SNMP works.

NWSNMP is an NLM that performs two major functions, as follows:

NWSNMP performs SNMP Get, GetNext, and Set requests on managed information, at the request of SNMP remote clients.

NWSNMP builds and sends SNMP traps on behalf of requesting entities.

NWSNMP performs operations only at the request of other software entities, such as remote clients or applications that make data available to it. It does not actively manage information.

NWSNMP is transport-independent. The current version of NWSNMP can receive and send SNMP information over the Internetwork Packet Exchange ™(IPX™), IP, and AppleTalk DDP protocols.

SNMP Basic Concepts

NWSNMP Concepts

Required Software

Obtaining RFCs and MIBs

Parent Topic:

NWSNMP: Guides

SNMP Basic Concepts

This chapter outlines the basic concepts and terminology used in Simple

Network Management Protocol (SNMP) in general. If you are already familiar with SNMP basic concepts and terminology, continue on to NWSNMP Concepts.

SNMP Introduction

Functional SNMP Components

Management Information Bases

Instrumentation Agents

Get, GetNext, and Set Functions

Parent Topic:

NWSNMP Introduction

NWSNMP Concepts

NWSNMP acts as an SNMP for objects that register with it. Once an object registers with NWSNMP, NWSNMP can respond to Get, GetNext, and Set requests on behalf of that object.

NWSNMP accesses data in abstract objects, but does not manage it. Your instrumented NLM™ maintains the data and provides functions by which NWSNMP accesses the data. NWSNMP receives requests, validates them, and then calls the functions you provide to actually manage the data. Thus, NWSNMP acts as a multiplexor and translator rather than as a data manager.

Instrumentation

NWSNMP Architecture

Object Registration and Deregistration

Operational Flow of an Instrumented NLM

MIBs

Access Control

Parent Topic:

NWSNMP Introduction

Functional SNMP Components

The SNMP model defines the following two components:

A **network manager**, which resides in the remote client

A **network manager** is an entity that can query the SNMP agents using SNMP operations. It provides a user interface, often graphical, that allows users to request data or see alarms resulting from traps. Often, it also archives data to permit trend analysis.

An **SNMP agent**, which resides in each manageable network device

An **SNMP agent** executes requests made by a network manager by gathering information from the network device in which the SNMP agent resides, and returning that information to the network manager.

These two components together communicate configuration and management needs over the network through either an IPX™ or an UDP protocol. The network manager sends configuration and management requests to the SNMP agent and the SNMP agent responds to those requests.

The SNMP agent and the network manager use other SNMP components that assist in the configuration and management of the network device.

Management Information Bases (MIBs), which reside on the remote client

A **MIB** is a text file that contains management and configuration information for a network device. MIBs provide the network manager with the information to make a request. MIBs are also used as the foundation for creating Instrumentation Agents (IAs) that reside on the network devices.

Instrumentation Agents (IAs), which reside on each manageable network device

An **IA** can be an application that contains management and configuration query information for the network device. An IA provides the SNMP agent with the information to respond to a network manager's request.

Where SNMP Components Reside

SNMP Operation

Parent Topic:

SNMP Basic Concepts

SNMP Operation

Network devices are managed through transactions between the network manager and the SNMP agent. SNMP provides two kinds of management transactions:

Request/Response transaction, a manager-request to SNMP agent-response

Trap transaction (unsolicited notification), an SNMP agent-to-manager transaction

The following sections describe these transactions.

Request/Response Transaction

Trap Transaction

Parent Topic:

Functional SNMP Components

Related Topics:

Where SNMP Components Reside

Management Information Bases

A Management Information Base (MIB) is a description of a set of manageable objects. A manageable network device can implement one or more MIBs, depending on its function. A MIB is similar to a database schema in that it describes both the structure and format of a set of data.

When a vendor develops an SNMP network device, the vendor typically designs a MIB that models the data important to the management of that type of network device. A vendor-specific type of MIB is termed a **proprietary** MIB because it is owned by the vendor. However, most vendors make their proprietary MIBs publicly available because they want their network devices to be manageable (see Obtaining MIBs).

This section describes the following information about MIBs:

The Two Uses of MIBs

The Global Naming Tree and ASN.1 Notation

Three Types of SNMP Objects

MIB Module Contents

Obtaining MIBs

Parent Topic:

SNMP Basic Concepts

Three Types of SNMP Objects

There are three types of objects in SNMP operations:

Scalar Objects are single instances of manageable objects.

Table Objects contain multiple instances of a grouping of scalar objects.

Group Objects contain a grouping of other objects, including other group objects, scalar objects, and table objects.

You can manipulate only scalar objects.

Parent Topic:

Management Information Bases

Related Topics:

MIB Module Contents

The Two Uses of MIBs

The Global Naming Tree and ASN.1 Notation

Obtaining MIBs

MIB Module Contents

This section describes the MIB. The textual representation of a MIB is a named module, written in a subset of the ASN.1 Notation language, which groups together related definitions. The MIB modules can be stored in an ASCII text file.

Note the following facts about the contents of a MIB module:

A comment is anything between a double hyphen and the end of the line or another double hyphen. For example:

```
-- This is a comment

-- This is a comment -- This is not part of the comment
```

A name is an arbitrary number (one or more) of letters, digits, and hyphens. A hyphen cannot be the last character; and a hyphen cannot be immediately followed by another hyphen.

An object name must start with a lowercase letter.

Type names and module names must start with an uppercase letter.

White space, such as (spaces, tabs, and new lines) are not significant except that names, keywords, and symbols, such as “::=” must not be interrupted by white space or comments.

The Sample MIB Module example shows the basic format of a MIB module.

Sample MIB Module

```
RFC1213-MIB DEFINITIONS ::= BEGINIMPORTS   mgmt, NetworkAddress, IpAddr
 DefinitionsEND
```

The following paragraphs explain the MIB module shown above:

`RFC 1213-MIB` (commonly known as MIB -II) is the name of the MIB module.

`IMPORTS` lists a collection of objects, types, and macros that are referenced in this module but are defined in the module named after the word `FROM`.

Definitions contains the actual object definitions (not shown). Four kinds of definitions can be found:

Value Assignments

Type Assignments and Textual Conventions

Manageable Object Definitions

Trap Definitions

Parent Topic:

Management Information Bases

Related Topics:

Obtaining MIBs

The Two Uses of MIBs

The Global Naming Tree and ASN.1 Notation

Three Types of SNMP Objects

Get, GetNext, and Set Functions

This section addresses the values the Get, GetNext, and Set requests return and how the SNMP agent handles the Get, GetNext, and Set requests.

Get and GetNext Handling

Set Handling

Parent Topic:

SNMP Basic Concepts

Instrumenting Objects

The NWSNMP includes an instrumentation interface and a MIB compiler that make the instrumentation of an object simple. You need to provide a Management Information Base (MIB) that describes the objects you are going to register with NWSNMP. Using your MIB, the NWSNMP MIB compiler produces C code that describes the manageable objects defined in your MIB. The module exporting the objects passes the resulting C structures to NWSNMP to register the objects. Thus, NWSNMP learns about the objects from this code.

Because NWSNMP now knows all about the objects, the effort of handling the object is reduced significantly. However, for this approach to work, you must use the MIB compiler and the instrumentation interface to provide NWSNMP with enough information about the objects so that it can make knowledgeable requests to read and manipulate the various objects in each MIB. Your code does this by providing a set of object-handling functions based on type definition functions (see the Specific Object-Handling Functions: Instrumentation Prototype Definitions table). The NWSNMP instrumentation interface defines a structure, the **Object Control Block**, to hold pointers to these functions.

Although the MIB compiler provides C code that describes your objects, it does not allocate space for the objects. You must provide structures for the object data and the code that maintains it. Such a structure allows you to organize the object data in a manner convenient to your code.

The instrumentation interface of NWSNMP does **not** handle the following tasks:

It does not eliminate the problems of values that change at interrupt level or objects that can only be accessed through blocking operations. In other words, a single SNMP action cannot be guaranteed to reflect a single point in time.

It does not handle any network management protocol other than SNMP.

It does not provide an interface for counting or for threshold detection. By its design, this interface deals at a level above where actual statistics are kept, as a result NWSNMP does not know where or how the actual values are maintained.

Consequently, manipulating values and detecting threshold crossing is left entirely to the instrumented code. This is consistent with the RFC specifications for SNMP, which provide no general mechanism to support thresholds.

Developing a Module that Exports an SNMP Object

Working with MIBs

Providing Instrumentation Functions

Registering an Object

Compiling and Linking the NLM

Universal Tasks Handled by NWSNMP

Parent Topic:

NWSNMP: Guides

Developing a Module that Exports an SNMP Object

Developing a module that exports an SNMP scalar, group, or table object requires the following steps:

1. Acquire and compile a MIB that describes the desired object. See Working with MIBs.
2. Provide the instrumentation functions required by NWSNMP for the Get, GetNext, and Set requests. There are two methods for this: standard object-handling functions and specific object-handling functions. These methods are compared in Providing Instrumentation Functions.
3. Write code to register the MIB with NWSNMP. See Registering an Object.
4. Compile and link the NLM™ file. See Compiling and Linking the NLM.

Parent Topic:

Instrumenting Objects

Working with MIBs

Before you can register an object with NWSNMP, you must provide a MIB definition of your object. The MIB describes the object in a manner that NWSNMP understands. Define your MIB by using the SNMP “Concise MIB Definitions” defined in RFC 1212. Once you have a MIB, use the MIB compiler to create both .C source and .H header files to include in your NLM™.

Abstract Objects

Hierarchical and Flat Structures

Acquiring a MIB Definition

Compiling a MIB

Parent Topic:

Instrumenting Objects

Providing Instrumentation Functions

Because NWSNMP knows all about your abstract object, the overhead of processing the object is reduced significantly. The NWSNMP includes two types of functions:

Standard Object-Handling Functions

Specific Object-Handling Functions

Parent Topic:

Instrumenting Objects

Related Topics:

Instrumentation Structures

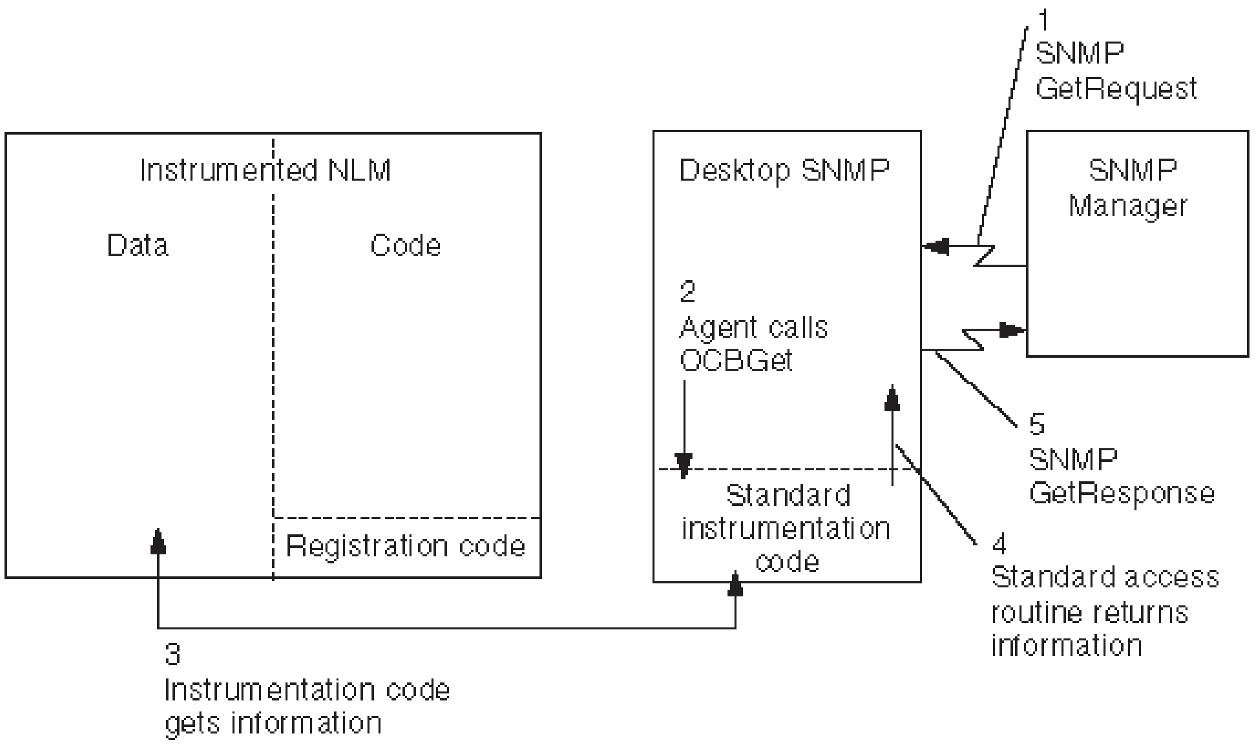
Standard Object-Handling Functions

For simple objects that can be set up as one- or two-dimensional tables, the NWSNMP provides standard object-handling functions. The idea behind these standard functions is that, in most cases, abstract objects have a simple organization that can be easily handled by standard support functions. Therefore, these access methods require a particular organization to the object data.

With the standard access methods, instead of writing your own access functions, you authorize the NWSNMP to use standard object-handling functions provided within NWSNMP code. You also pass a pointer to your data to NWSNMP when you register your MIB.

NWSNMP provides a library of standard object-handling functions for handling various standard data organizations. Thus, NWSNMP directly accesses the information you implement with standard access methods (see the following figure).

Figure 3. Get with Standard Object-Handling Methods



1. An SNMP manager sends a GetRequest Protocol Data Unit (PDU) to the NWSNMP.

2. NWSNMP parses the request and calls its standard **OCBGet** function.

3. The **OCBGet** function uses location information you provide to get the data from the MIB data you maintain within your code.

4. The **OCBGet** function returns the information to NWSNMP.

5. NWSNMP encodes a GetResponse PDU with the information and returns it to the SNMP manager that made the request.

Several typical examples of abstract objects can be handled simply and efficiently. For example, one common organization is a statistics table occupying an array of unsigned long INTEGERS indexed by attribute numbers. NWSNMP provides the various handling functions to deal with these simple organizations of attributes.

If you can organize your data into one of the standard formats, detailed in Instrumentation Using Standard Object-Handling Functions the job of instrumenting the object becomes much simpler. Instead of writing specific functions for Get, GetNext, and Set requests, you need only add a macro call to define an Object Control Block for each object in the MIB. NWSNMP provides the necessary functions for the operations.

Instrumentation Using Standard Object-Handling Functions

Parent Topic:

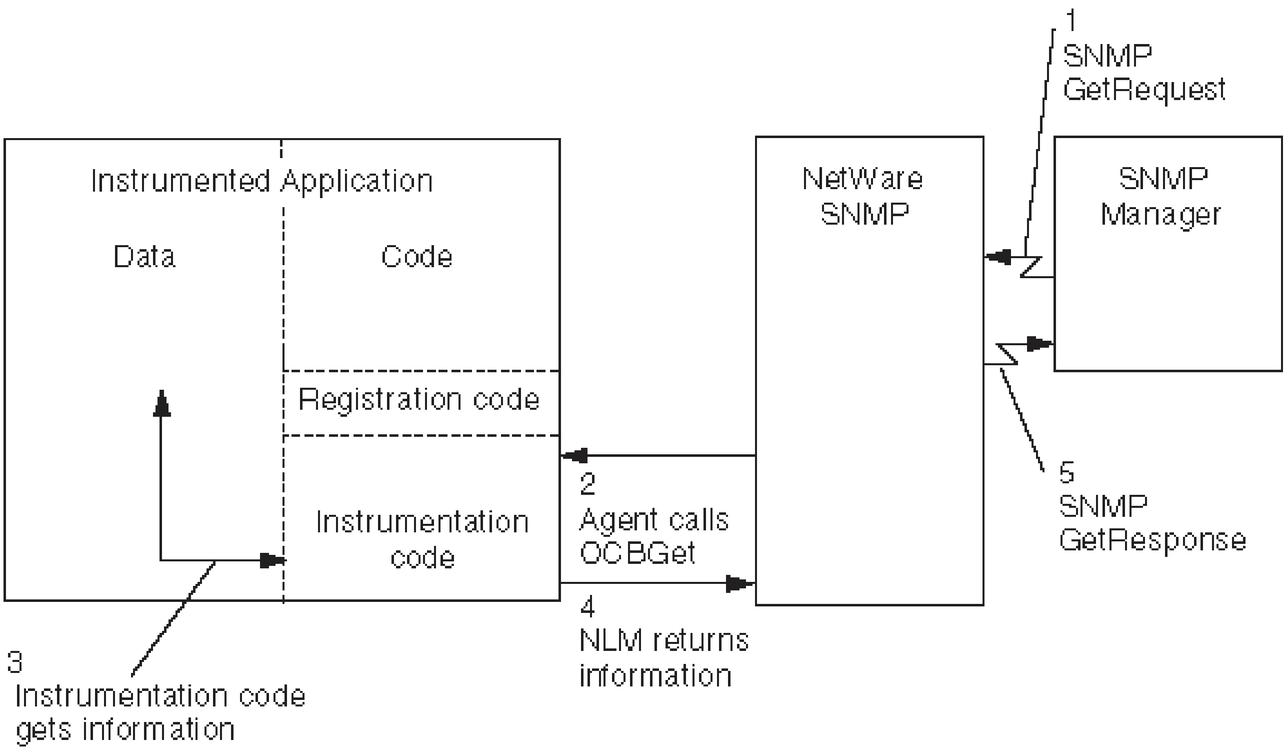
Providing Instrumentation Functions

Specific Object-Handling Functions

If your NLM™ provides specific object-handling functions for an abstract object instead of using the standard ones, NWSNMP never accesses your data directly. Although NWSNMP knows the type of data you have placed in an abstract object and whether the data is writable, NWSNMP does **not** know where the data is or how to read it. NWSNMP simply uses the functions you provide to access your data in response to requests from other management agents. This means you must provide specific functions (Instrumentation Code) that NWSNMP calls in response to Get, GetNext, and Set requests that affect your object. These functions are discussed in Instrumentation Using Specific Object-Handling Functions.

The following figure shows how NWSNMP handles a Get request against an instrumented object by calling the function based on **OCBGet** that is provided by the instrumented NLM.

Figure 4. Get with Object-specific Instrumentation Functions



1. An SNMP manager sends a GetRequest Protocol Data Unit (PDU) to NWSNMP.

2. NWSNMP parses the request and calls the **OCBGet** function for your object.

3. Your application receives the request and does whatever processing is required to obtain the information.

4. Your application returns the information to NWSNMP.

5. NWSNMP encodes a GetResponse PDU with the information and returns it to the SNMP manager that made the request.

Instrumentation Using Specific Object-Handling Functions

Parent Topic:

Providing Instrumentation Functions

Instrumentation Structures

The instrumentation interface uses the following data structures to describe each instrumented object:

Object Information Block

A structure, ObjInfo, that describes the object to NWSNMP. The Object Information Block is generated by the MIB compiler and used internally by NWSNMP. **Do not** modify the Object Information Block once it has been created.

Object Control Block

A structure, ObjControl, that provides pointers to the operations that can be applied to an abstract object when NWSNMP responds to a request for a Get, GetNext, or Set request. YourNLM™ must define the necessary functions (unless you are using the standard object-handling methods), define the Object Control Block, and place pointers to the functions in the block. NWSNMP uses the information from the block to perform operations on your abstract object.

Attribute Information Block

A structure, AttrInfo, that describes the characteristics of a particular attribute. The Attribute Value Block points to this block to provide the accesss functions with information about the attribute being accessed. The information in this block is derived from the compiled MIB. Do not modify the Attribute Information Block once it has been created.

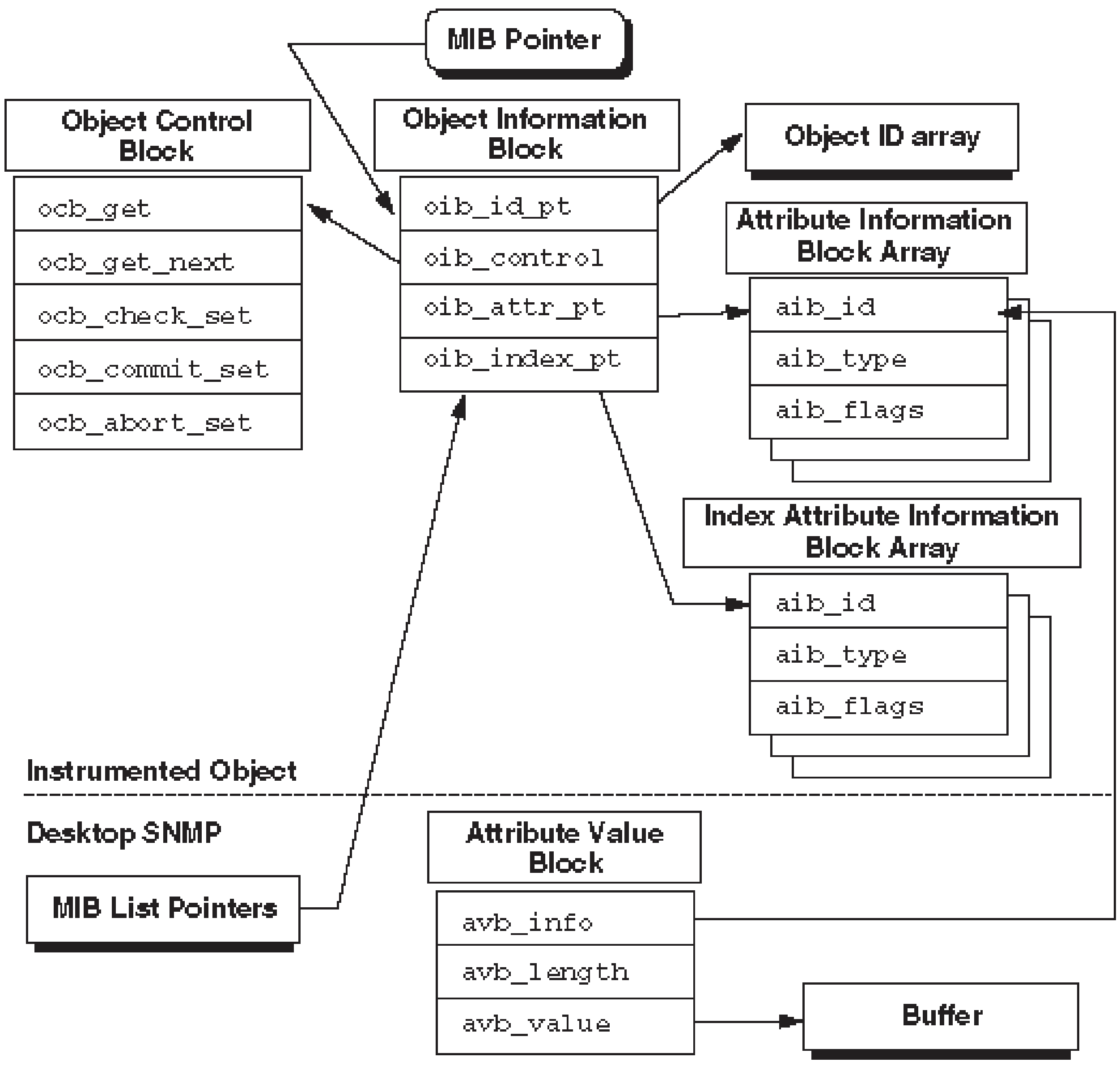
NOTE: When discussing an Attribute Information Block, an attribute refers to an object in a table or group.

Attribute Value Block

A structure, AttrValue, that passes information between NWSNMP and an instrumented NLM during Get, GetNext, and Set requests about the attribute being accessed.

The following figure shows the relationships among these structures. For simplicity, it does not show all the fields in the Attribute Information Block or Attribute Value Block structures.

Figure 5. Instrumentation Structures Figure



Object Information Blocks

Object Control Blocks

Attribute Information Blocks

Attribute Value Blocks

Parent Topic:

Providing Instrumentation Functions

Related Topics:

Standard Object-Handling Functions

Specific Object-Handling Functions

Universal Tasks Handled by NWSNMP

To minimize the effort on the part of the exporting modules, NWSNMP takes care of as much of the processing of SNMP requests as possible. It does this using the information from the MIB, such as the types of the values of attributes and indexes. Thus, NWSNMP can handle the following types of tasks:

Grouping

Data Conversion

Time Field Conversion

OCTET STRING Conversion

Abstract Objects with Exactly One Instance

Attribute Range Checking

Parent Topic:

Instrumenting Objects

NWSNMP Client Interface

NWSNMP can be viewed as a general purpose data repository. Such a repository is an important and useful resource, so NWSNMP provides a client interface that allows other local processes to access its data store. For example, agents for other network management protocols can access values through NWSNMP.

The NWSNMP client interface provides the following functions that allow any NLM™ file running on the same server to manipulate the NWSNMP data store:

Table . NWSNMP client interface functions

Function	Purpose
SACReadAttributes	Returns the values of a list of attributes (SNMP Get), given an abstract object ID and instance index information.
SACSetAttributes	Sets the indicated attributes to particular values (SNMP Set), given an abstract object ID and instance index information.

In no case is ownership of memory passed across the interface, so all pointers can point to temporary strorage that is discarded after the call returns. The functions and associated structures are described inNWSNMP Functions and Structures.

The following covers the values these functions return and the basics of how NWSNMP behaves when handling the Get, GetNext, and Set requests, whether received as strict SNMP requests or as requests that use the NWSNMP client interface functions.

Client Interface Return Values

NWSNMP Get Handling

NWSNMP Set Handling

Parent Topic:

NWSNMP: Guides

NWSNMP Functions and Structures

This section describes the functions and structures that make up NWSNMP.

The functions are named beginning with acronyms to indicate their purpose. The acronyms are as follows:

OCB	"Object Control Block" instrumentation. These functions give the structure for the calls your NLM™ file must provide for each abstract object. NWSNMP uses these functions to fulfill management requests.
SAC	"SNMP agent client" instrumentation. These functions allow you to Get and Set local management information to which NWSNMP has access.
SAI	"SNMP agent instrumentation".

	These functions allow you to register and deregister management information bases (MIBs).

The following sections list the functions and structures by interface type:

Instrumentation Functions

Instrumentation Structures

Client Interface Functions

Trap Interface Functions

Client and Trap Structures

Parent Topic:

NWSNMP: Guides

NWSNMP: Tasks

Compiling and Linking the NLM

1. **Once you have developed the instrumentation and registration functions, compile all modules as you normally do, including the C module created by the MIB compiler. Remember to include the header file output from the MIB compiler in the appropriate source modules.**

2. **After compiling, link the NLM™. The NLM does not need to export any symbols to support a MIB. When linking, import the SNMP entry points by including AGENT.IMP in the IMPORT section of the linker command file.**

Parent Topic:

Instrumenting Objects

Configuring NWSNMP to Provide MIB-II Information

You can set several MIB-II variables in the SYS:\ETC\SNMP.CPG file, as follows:

sysName	Gives the system name.
sysLocation	Gives the system location.
sysContact	Gives the name of the system administrator or other person who should be contacted about system problems or maintenance.

1. **To set these varables, edit the SYS:\ETC\SNMP.CFG file with any ASCII text editor and follow the instructions given in the file comments.**

Parent Topic:

NWSNMP: Guides

Configuring NWSNMP to Send Traps to Your

Application

1. **To receive traps sent by NWSNMP, make sure your remote client address is listed in the IP or IPX™ section of the SYS:\ETC\TRAPTARG.CFG file. Edit the file with any ASCII text editor and follow the instructions given in the file comments.**

Parent Topic:

NWSNMP: Guides

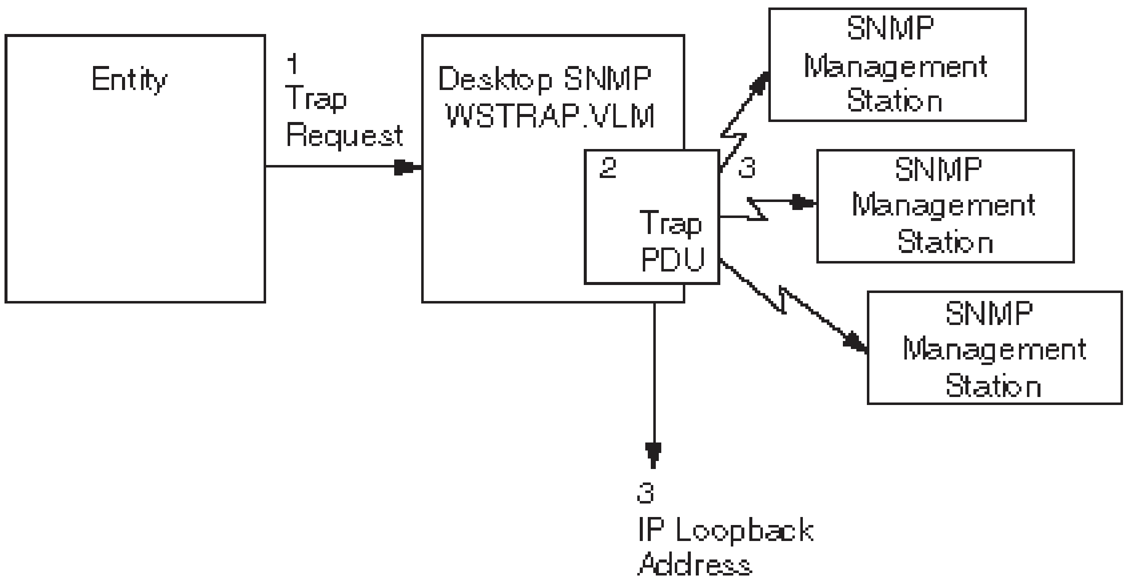
Initiating Traps

Applications can initiate SNMP Traps, use NWSNMP to build the trap Protocol Data Units (PDUs), and send them to remote clients (management stations) that are configured to receive traps. The trap initiation process involves the following three steps:

1. **An entity initiates a trap by calling the trap function provided by NWSNMP.**
2. **NWSNMP builds a trap PDU based on the information the entity provides.**
3. **NWSNMP sends the trap PDU to all Internet Packet (IP) and Internetwork Packet Exchange™IPX™ remote clients listed in the SYS:\ETC\TRAPTARG.CFG file.**

   NOTE: When NWSNMP is installed, the IP loopback address is the only address listed in the SYS:\ETC\TRAPTARG.CFG file.

Figure 6. Trap Initiation



Any entity can call NWSNMP to send traps. The entity need not be one that manages an instrumented object to send traps. Because of the way the process works, the entity that requests the trap need not know which remote clients receive the trap.

NWSNMP provides the following functions for trap initiation and data retrieval. Use these functions to provide NWSNMP with the necessary information to create an SNMP trap PDU:

Table . Trap initiation and data retreival functions

Function	Purpose
SACTrap	Initiates an SNMP trap. This is generally used only for enterprise-specific traps.
SACReadSNMPObject	Reads an SNMP object value given an SNMP object ID. An entity can use this function to

	retrieve an object value for use in a trap initiation request, if the entity is unable to monitor the value directly.

These functions are defined in NWSNMP Functions and Structures.

In order for SNMP remote clients to interpret an SNMP trap, the trap must be defined in a trap MIB. Trap MIBs must be defined according to RFC-1215, "A Convention for Defining Traps for use with the SNMP." Your trap MIB must be compiled on each remote client that receives traps from your application before users at that remote client can interpret the trap.

Parent Topic:

NWSNMP: Guides

Loading NWSNMP

NWSNMP loads automatically when your server starts. However, you can explicitly start NWSNMP with the **LOAD** command, or reenter the **LOAD** command while NWSNMP is running, to change the options. You can also modify the **LOAD SNMP** line in the AUTOEXEC.NCF file to include your preferred default options.

1. **To load NWSNMP, use the following command format:**

```
load snmp [options] <Enter>
```

The options allow you to establish the community name used in SNMP traps. NWSNMP also provides default community names (see Community Name Options) for the monitor (read-only) and control (read/write) communities. NWSNMP uses these names for access control. The community name contained in a request message from an SNMP remote client must match the name established by NWSNMP.

NOTE: If NWSNMP receives a request Protocol Data Unit (PDU) whose community name is not authorized, NWSNMP does not respond to the request. For example, suppose the control community name is secret, and NWSNMP receives a SetRequest PDU with a community name of public. NWSNMP discards the SetRequest PDU and does not respond.

Community types can also be disabled. When a community type is disabled, no management entity can access information for that community. For example, if you disable the control community, no one can use NWSNMP to do **Set** operations against the data NWSNMP manages.

Parent Topic:

NWSNMP: Guides

Related Topics:

TCPIP and NWSNMP Load Information

Registering an Object

The MIB compiler generates a C file that describes the MIB being compiled. That C file exports a single symbol: the name of the MIB, as defined by the DEFINITIONS keyword in the concise MIB definition. Hyphens, which are illegal in C identifiers, are changed to underbars. For MIB-II, for example, the symbol is RFC1213_MIB.

1. **During initialization, the exporting NLM™ must pass this handle (the name of the MIB), along with a MIB resource tag, to NWSNMP using SAIRegisterMIB. The resource tag is not used in NWSNMP and it must be set to zero. This call registers the entire MIB tree with NWSNMP.**

Parent Topic:

Instrumenting Objects

NWSNMP: Concepts

Abstract Objects

In the SNMP, each object is identified with an SNMP object identifier, which describes the object location in the global naming tree (see The Global Naming Tree and ASN.1 Notation). SNMP standards specify that an identifier is treated as a single, indivisible value. For the purposes of the NWSNMP interface, however, the object identifier is considered to have three parts: an abstract object identifier, an attribute identifier, and an instance identifier.

> **IMPORTANT:** The concept of an abstract object is particular to the NWSNMP instrumentation interface. Although this interface makes use of this concept, it is not part of any SNMP standard. In fact, SNMP avoids such divisions of the object identifier in the protocol itself. This section is essential for understanding how the NWSNMP intrumentation interface works, but should not be mistaken for the discussion about SNMP practice.

Sub-IDs at the beginning of the SNMP object ID identify an **abstract object**. An abstract object is an object in the global naming tree that has at least one node below it in the tree.

A single sub-ID **attribute identifier** follows the abstract object ID, separating it from the instance ID. In MIB definitions, this sub-ID is always the last sub-ID of an object with no subtrees.

An **instance identifier** at the end of the SNMP object ID identifies which particular instance of the object is being requested. In MIB definitions, this information is identified by the INDEX clause.

In the NWSNMP instrumentation interface, the abstract object ID and the instance ID find a specific object instance. Each object instance is then viewed as a row of values indexed by the attribute ID.

The following sections give two examples, taken from MIB-II.

A Simple Example: ipForwarding

One SNMP object from the standard IP section of MIB-II is `ipForwarding`. Its official SNMP object ID is 1.3.6.1.2.1.4.1.0. The string 1.3.6.1.2.1.4 is the abstract object ID, identifying the hypothetical abstract object **IP statistics**. The 1 following the abstract object ID identifies the attribute **forwarded packet count**. The trailing 0 is the instance identifier. There is never more than one instance of `ipForwarding`, so the instance identifier is always zero.

A More Complex Example: ipRouteMetric1

Another SNMP object from MIB-II is ipRouteMetric in the IP routing table (`ipRoutingTable`). Its object ID is 1.3.6.1.2.1.4.21.1.3.d.d.d.d. Here the abstract object ID is 1.3.6.1.2.1.4.21.1, which identifies an abstract routing entry. The 3 gives the attribute ID, which in this case refers to **the first routing metric**. The d.d.d.d is the instance ID. For routing entries, these four sub-IDs are the IP address of the entry.

Overlap Between Abstract Objects

Because SNMP objects do not formally have this format, sometimes abstract objects inconveniently overlap. For example, the preceding examples show that `ipRoutingTable`, of which `ipRoutingMetric1` is an attribute, actually occurs as attribute 21 in the abstract object **IP statistics**, whose abstract object ID is 1.3.6.1.2.1.4. Although this does complicate the object lookup code within NWSNMP (particularly the implementation of GetNext), the NLM™ exporting the objects is not affected and, in fact, does not realize this overlap has occurred.

Parent Topic:

Working with MIBs

Related Topics:

Hierarchical and Flat Structures

Acquiring a MIB Definition

Compiling a MIB

Abstract Objects with Exactly One Instance

NWSNMP understands abstract objects that have only a single instance. When these objects are requested, NWSNMP ensures that .0 is always the instance ID. In addition, because GetNext is a trivial request for such objects, you do not need to provide an **OCBGetNext** function when instrumenting the object. NWSNMP uses **OCBGet** instead.

Parent Topic:

Universal Tasks Handled by NWSNMP

Related Topics:

Grouping

Data Conversion

Time Field Conversion

OCTET STRING Conversion

Attribute Range Checking

Access Control

At the lowest level, MIB definitions indicate whether a particular attribute is readable or writable. NWSNMP enforces these restrictions. However, these characteristics merely indicate whether it makes sense to ever read or write a particular value.

NWSNMP provides control through access lists for two community types, monitor (read-only) and control (read/write). If community names are defined for these types, Get, GetNext, and Set requests must supply the appropriate community name. If the community name is not valid, NWSNMP discards the request and performs no further processing. NWSNMP also includes a community name in the traps it builds. For information about defining the community names NWSNMP uses, see Community Name Options.

NWSNMP handles all access control on behalf of instrumented objects. Because NWSNMP discards any requests that do not have valid access, **without passing them to the instrumented object**, the instrumented code is not aware of the access control facilities in place.

Parent Topic:

NWSNMP Concepts

Acquiring a MIB Definition

The first step in instrumenting an object is to obtain a MIB definition for the object in the "Concise MIB Definitions" format described in RFC 1212.

For many common objects, you can obtain a MIB definition created by the Internet Engineering Task Force (IETF) or some other standards body. In some cases, however, the object might not have been previously defined in an Internet document. In this case, you might need to seek out an unpublished definition, or write and publish your own definition.

When creating a MIB of your own, focus on the information to be stored in the MIB. The MIB must be well-structured. Objects that are related to each other must be placed together into one group. The best way to create a MIB of your own is to modify an existing one. You can modify one of the MIBs provided in the NetWare SDK or use a public MIB.

> **NOTE:** You must include an INDEX clause for any tabular (multiple-instance) object defined in your MIB. The NWSNMP instrumentation interface uses the INDEX clause to locate the instrumented object.

If you are using an unpublished MIB definition, it is recommended that you send the MIB into the IETF review process for eventual publication. SNMP is an open protocol, and MIBs used by your products must be available to all parties, encouraging them to support the objects your products export.

Once you decide the types of objects your MIB will contain, you can decide on the data types of the attributes. NWSNMP supports the ASN.1 Notation data types defined in the Structure of Management Information (SMI), RFC 1155. These types, along with their internal C representations, are listed in the following table. Use only the data types that reduce to these primitives in your MIB.

Table . Supported ASN.1 Notation Data Types

MIB Type	**Examples**	**C Type**	**Description**
INTEGERS	ipInReceives	unsigned long	Includes `GAUGE` and `COUNTER`.
short OCTET STRINGS	ipRouteDest ipAddress	unsigned long	Fixed length, four octets or less. The first octet goes into the low byte order, and so on. NWSNMP distinguishes between short and long OCTET STRING types. (SNMP does not make this distinction.)
long OCTET STRINGS	sysName DisplayString physAddress	unsigned char *	Maximum length greater than four octets; length can vary. Passed by reference using a pointer to an array. NWSNMP distinguishes between short and long OCTET STRING types. (SNMP does not make this distinction.)
OBJECT ID	sysObjectID	unsigned long *	Pointer to an array; passed by reference.
CHOICE	atNetAddress	unsigned long	The only `CHOICE` supported is `atNetAddress`; it is handled like an `IpAddress`.

Parent Topic:

Working with MIBs

Related Topics:

Compiling a MIB

Abstract Objects

Hierarchical and Flat Structures

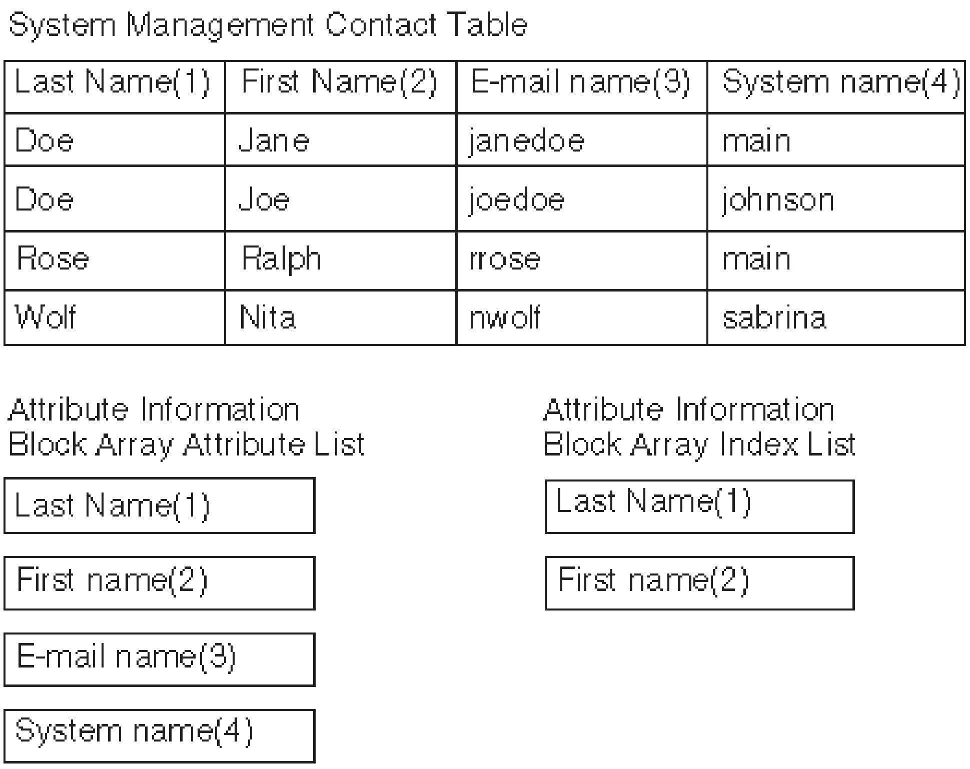
Attribute Information Blocks

Each Attribute Information Block describes a single attribute maintained by the abstract object. It provides the remainder of the MIB tree identifier for that attribute, the data type and ASN.1 Notation type of the attribute, the minimum and maximum sizes of the attribute, and flags that describe other characteristics of the attribute.

The Object Information Block points to two arrays of Attribute Information Blocks. One array describes every attribute in the table or group, while the other describes only the attributes used to index object instances. The blocks in the indexing array are always given in exactly the same order as the attributes listed in the INDEX clause of the object MIB.

For example, suppose you are instrumenting a table that contains names of system management contacts. The table contains the following columns: last name, first name, electronic mail (E-mail) name, and primary system name. To look up a manager uniquely, you need the first and last names. The following figure shows a sample table and shows which Attribute Information Blocks are contained in its attribute list and index list arrays.

Figure 7. Sample Attribute Information Block



System Management Contact Table

Last Name(1)	First Name(2)	E-mail name(3)	System name(4)
Doe	Jane	janedoe	main
Doe	Joe	joedoe	johnson
Rose	Ralph	rrose	main
Wolf	Nita	nwolf	sabrina

Attribute Information Block Array Attribute List

- Last Name(1)
- First name(2)
- E-mail name(3)
- System name(4)

Attribute Information Block Array Index List

- Last Name(1)
- First name(2)

Parent Topic:

Instrumentation Structures

Related Topics:

Attribute Value Blocks

Object Information Blocks

Object Control Blocks

Attribute Range Checking

NWSNMP remembers which attributes are legal for each abstract object from a registered MIB, and ensures that the attribute requested is legal. An object handler might want to perform its own attribute range checking if it does not fully support the attribute ranges defined in its associated MIB.

Parent Topic:

Universal Tasks Handled by NWSNMP

Related Topics:

Grouping

Data Conversion

Time Field Conversion

OCTET STRING Conversion

Abstract Objects with Exactly One Instance

Attribute Value Blocks

NWSNMP deals frequently with attribute values. NWSNMP expresses a group of attribute values as a null-terminated linked list of Attribute Value Blocks. Each of these blocks contains either an attribute value or a description of a buffer for receiving an attribute value. The Attribute Value Block points to an Attribute Information Block that describes the attribute contained in it. Thus, the Attribute Information Block allows an entity to handle an Attribute Value Block in a general way without knowing its specific contents. Entities that must know more about the attribute can look in the Attribute Information Block.

The length field of the Attribute Value Block has two uses:

For attributes whose values fit into an unsigned long field, the value is passed in the *avb_length* field. These attributes are identified by the AIF_IMMEDIATE flag in the Attribute Information Block, and are known as **immediate** attributes.

For all other attributes, the value is passed in a buffer to which the *avb_value* field points. For these values, the buffer, the *avb_value* field, and the *avb_length* field are used as follows:

When NWSNMP creates an Attribute Information Block for a Get or GetNext request, it allocates the buffer, places a pointer to the buffer in avb_value, and places the length, in bytes, of the buffer in *avb_length*. When the responding NLM places a value in the buffer, it also must replace the value in *avb_length* with the actual length of the value, in bytes.

When NWSNMP creates an Attribute Information Block for a Set request, it allocates the buffer, puts the new attribute value in the buffer, and places the actual length of the value, in bytes, in *avb_length*.

Parent Topic:

Instrumentation Structures

Related Topics:

Object Information Blocks

Object Control Blocks

Attribute Information Blocks

Client and Trap Structures

These structures provide information used by the client interface and trap interface functions:

TLV	Contains an ASN.1 type, length, and value; also used to describe a buffer for receiving such a value.
VarBind	Contains a single SNMP variable binding.

Parent Topic:

NWSNMP Functions and Structures

Related Topics:

Client Interface Functions

Trap Interface Functions

Client Interface Functions

The following functions allow SNMP Set, GetNext, and Get requests on local objects that NWSNMP controls:

SACReadSNMPObject	Reads an SNMP object value.
SACSetAttributes	Sets attributes in an abstract object.
SACReadAttributes	Reads attributes from an abstract object.

Parent Topic:

NWSNMP Functions and Structures

Related Topics:

Client and Trap Structures

Trap Interface Functions

Client Interface Return Values

The following table shows the values returned by the client interface functions. These values are the same as those used by the instrumentation interface functions.

Table . Return Values for the Client Inferface Functions

Return Value	Returned By	Description
SNMP_ERR_NOERROR	All	No error; the function succeeded.
SNMP_ERR_TOOBIG	**SACReadAttributes**	The response exceeds the space allocated to it.
SNMP_ERR_NOSUCHNAME	All	The indicated item does not exist. This must be returned if the instance ID indicates an instance that does not exist or if the object handler encounters an unrecognized attribute. In the latter case, the Attribute Value Block must also be flagged.

SNMP_ERR_BADVALUE	**SACSetAttributes**	Returned on a Set request if the object handler detects an attempt to set an attribute to an invalid value.
SNMP_ERR_GENERR	All	General error not covered by any of the other error codes. For example, a memory allocation failure.

Parent Topic:

NWSNMP Client Interface

Related Topics:

NWSNMP Get Handling

NWSNMP Set Handling

Community Name Options

The LOAD command line accepts three SNMP option parameters, as follows:

MonitorCommunity	Describes the read-only community (the community that is allowed to do Get and GetNext requests). The default value is public.
ControlCommunity	Describes the read/write community (the community that is allowed to do Set requests). Any community name established for read/write access is also valid for read-only access. By default, this community is disabled. When the control community is disabled, all write access is disabled.
TrapCommunity	Describes the community name used for traps. The default value is public.

NOTE: If the trap community name is disabled, NWSNMP **does not send traps**.

The syntax for the options is as follows:

```
M[onitorCommunity] [= [communityName] ]
```

```
C[ontrolCommunity] [= [communityName] ]
T[rapCommunity] [= [communityName] ]
```

The option parameters (such as MonitorCommunity) are not case-sensitive. In addition, when specifying option parameters, you need enter only the first character of the option name, although complete or partial names are also accepted. For example, T, TrapCom, and Trap are all interpreted as TrapCommunity.

The *communityName* is an arbitrary ASCII, case-sensitive string of up to 32 characters. It can include any characters except space, tab, open square bracket ([), equal sign (=), colon (:), semicolon (;), or number sign (#).

> **NOTE:** Although the option names are not case-sensitive, community names are case-sensitive. Thus, the names Public, public, and PUBLIC denote three different communities.

Enabling Access by a Single Community Name

To enable access to a community for a single community name, enter the option parameter, followed by an equal sign (=), followed by the community name. Thereafter, the community name offered by the SNMP remote client must match the specified value, or NWSNMP denies access for the request.

Enabling Access by a Single Community Name

If you follow the option name only by an equal sign with no argument, NWSNMP accepts **any** community name offered by an SNMP remote client for that community.

Disabling Access to a Community

To disable access to a community, enter the associated option name without following it by an equal sign (=).

Examples

To set the read/write community name to secret, use the following command:

```
LOAD SNMP ControlCommunity =secret <Enter>
```

To disable all read/write access, use the following command:

```
LOAD SNMP ControlCommunity <Enter>
```

To allow any community name to be used for read access, use the following command:

```
LOAD SNMP MonitorCommunity = <Enter>
```

To allow any community name to have read-only access and set the read/write community name to private, use the following command (note the abbreviated option names):

```
LOAD SNMP M = C =private <Enter>
```

To set the community name for traps to AgentTrap, use the following command:

```
LOAD SNMP TrapCommunity =AgentTrap <Enter>
```

Parent Topic:

Loading NWSNMP

Compiling a MIB

NWSNMP includes a MIB compiler, MIBCOMP.EXE, to compile your MIB. When the compiler processes your MIB definition, it creates a C module (.C) containing the C structures required by the NWSNMP to understand the MIB. It also creates a header file (.H) containing the C extern definitions for the data structures that the resulting C module is expecting from other C modules in the NLM™. The header file also contains #define statements for the object attribute IDs. Include the header file in the C modules that satisfy the extern definitions. This ensures that the structures provided agree with the requirements of the MIB definition.

IMPORTANT: The current version of the MIB compiler does not ignore introductory text, page headers, or page footers in standard RFC format MIBs. Therefore, you must edit your MIB definition to remove any text other than the actual RFC-1212 format MIB definition. This includes all text before the DEFINITIONS ::= BEGIN line, all headers and footers within the actual definition, and any text following the MIB END statement.

Parent Topic:

Working with MIBs

Related Topics:

Abstract Objects

Hierarchical and Flat Structures

Acquiring a MIB Definition

Data Conversion

The MIB compiler extracts data type information from the MIB. Using this information, NWSNMP converts instance IDs and attribute values between ASN.1 format, which SNMP uses in its packets, and the native format. This means data type information is built into NWSNMP and the MIB compiler, so you need to be concerned only with the native format.

Parent Topic:

Universal Tasks Handled by NWSNMP

Related Topics:

Grouping

Time Field Conversion

OCTET STRING Conversion

Abstract Objects with Exactly One Instance

Attribute Range Checking

Get and GetNext Handling

The Get and GetNext requests are very similar. As the SNMP agent processes a Get or GetNext request, it performs the following actions:

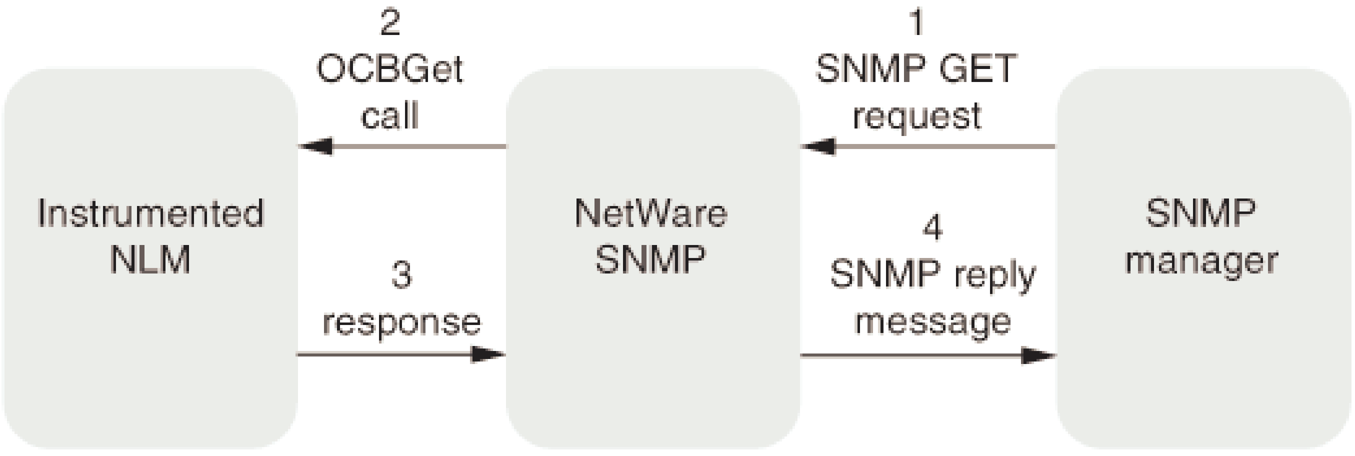
1. The SNMP agent decodes the Protocol Data Unit (PDU) request and prepares to get the requested information. In the process, it completes the following steps:
    a. The SNMP agent identifies the authority for the request (by way of the community name and any other authorization features implemented) and the SNMP objects being requested.
    b. The SNMP agent looks up each SNMP object requested in the data store of registered abstract objects. From the abstract object information, the SNMP agent identifies the abstract object ID, the attribute ID, and the instance ID. If the request does not have the authority to read this object, the SNMP agent rejects the request.
    c. If the request does have the proper authority, the SNMP agent groups all requests for each object ID/instance ID pair so they can be treated as a single request for information about the indicated object instance.
    d. For each object instance, the SNMP agent gets the data type of the object data.
    e. The SNMP agent converts the instance ID into the native format of the machine to which the ID arrives (for example, big endian). The request fails if the index is invalid: too long, too short, out of range, and so on.
2. If the decode process has been successful, the SNMP agent calls the appropriate **OCBGet** or **OCBGetNext** function within the Instrumentation Agent for the object, passing the Object Information Block, the instance ID, and the list of attributes to be read.
3. The SNMP agent receives the requested information from the

instrumented object.

4. Using the information returned by the **OCBGet** or the **OCBGetNext** function, the SNMP agent builds and sends a PDU response message, converting each piece of object information from the native format of the machine into the appropriate ASN.1 Notation format.

The following figure shows the Get and GetNext handling process.

Figure 8. Get and GetNext Handling Process Flow



Parent Topic:

Get, GetNext, and Set Functions

Related Topics:

Set Handling

Group Objects

A group object is a collection of individual objects. A group is similar to a directory that contains files. A MIB usually contains a number of group objects; group objects can contain a number of scalar objects, table objects, or other group objects.

In the Global Naming Tree example (see The Global Naming Tree and ASN.1 Notation), `mib-2(1)` is the node name of the MIB, which is actually a group object itself. The following groups are under `mib-2(1)`.

system(1)
interfaces(2)
at(3)
ip(4)

icmp(5)
tcp(6)
udp(7)
egp(8)
transmission(10)
snmp(11)
rmon(16)

In the Global Naming Tree example, the group tcp is expanded to show a scalar object and a table object. If every group under `mib-2(1)` were expanded, an object list similar to tcp would appear containing scalar and table objects.

Parent Topic:

Three Types of SNMP Objects

Related Topics:

Scalar Objects

Table Objects

Grouping

Management entities can frequently provide better service when they receive all attribute information for a given object instance simultaneously. To facilitate this, NWSNMP gathers all attributes requested for each object instance in the request into a list of information and passes it to the handler at the same time. For instance, a Get request might contain a list of object variables to be retrieved, instead of multiple Get requests with one variable per call.

Parent Topic:

Universal Tasks Handled by NWSNMP

Related Topics:

Data Conversion

Time Field Conversion

OCTET STRING Conversion

Abstract Objects with Exactly One Instance

Attribute Range Checking

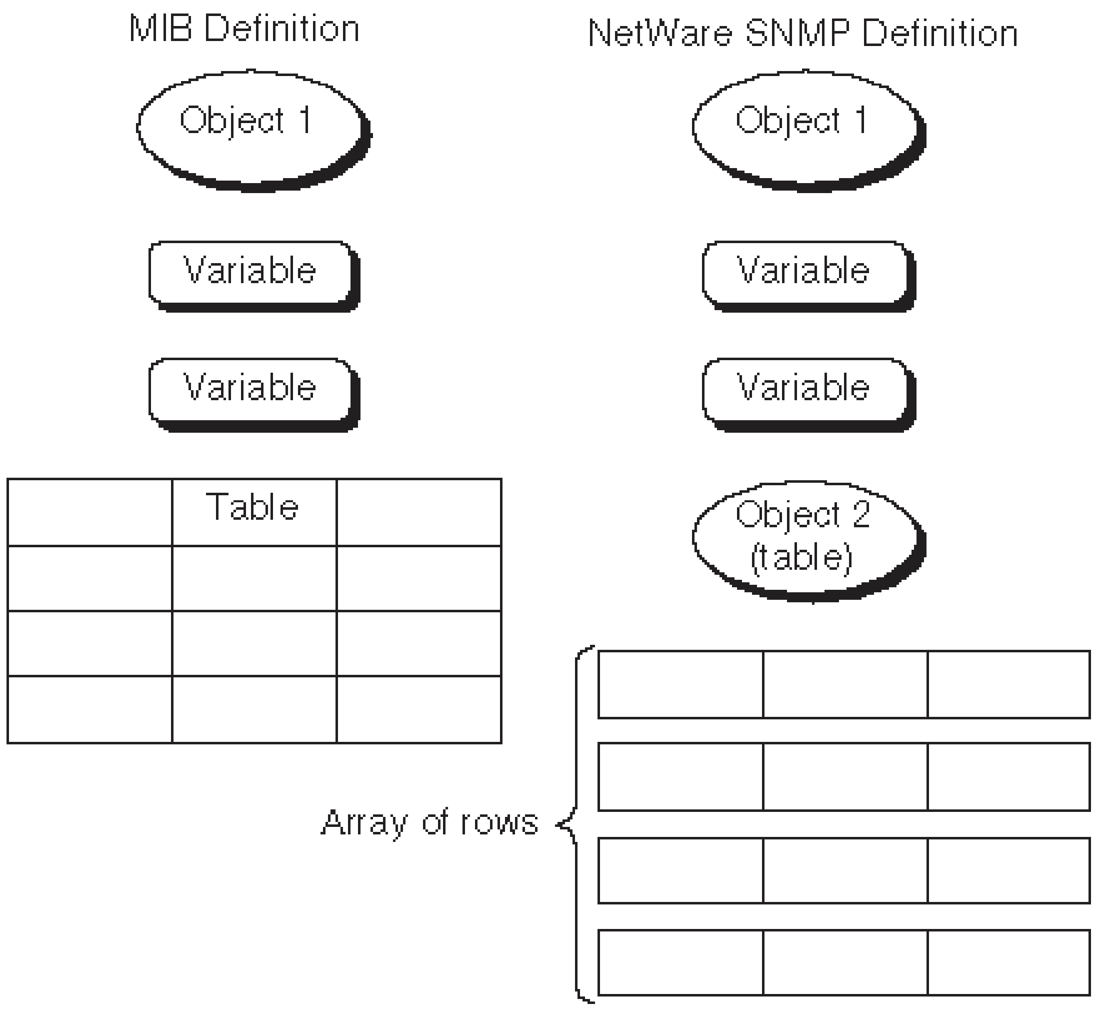
Hierarchical and Flat Structures

MIBs define a hierarchical structure of nested objects. For example, a MIB might define an object with two variables and a log table with three columns and four rows.

In contrast, the NWSNMP MIB compiler views objects in a flat, object-oriented structure. Thus, in the example above, the MIB compiler defines two objects: one object with two attributes to represent the two simple variables, and a second object to represent the table (see the following figure). The table is redefined as a series of four arrays, each with three elements.

This definition does not change the actual organization of the data. It simply decouples the hierarchical dependencies of the table and the remaining object attributes.

Figure 9. MIB Interpretation



Parent Topic:

Working with MIBs

Related Topics:

Acquiring a MIB Definition

Compiling a MIB

Abstract Objects

Instrumentation

Instrumentation is the process of making NLM™ object data available to NWSNMP. Thus, an **instrumented object** is one that has been made accessible to NWSNMP. Once an object is instrumented, its associated data is available to NWSNMP. NWSNMP can do Get and Set requests against the object data at the request of management entities.

Instrumenting an object to NWSNMP allows you to make the object manageable by SNMP without providing direct SNMP capabilities in your code. Instead, NWSNMP handles tasks related to creating SNMP Protocol Data Units (PDUs) and responding to requests from SNMP remote clients. This approach allows your NLM to ignore most of the peculiarities of SNMP and MIBs. In addition, this approach allows you to work at a higher level of abstraction than SNMP objects.

NWSNMP provides an interface for instrumenting objects to NWSNMP, as described in Instrumenting Objects.

Parent Topic:

NWSNMP Concepts

Related Topics:

Management Information Bases

Instrumentation Agents

Instrumentation Agents (IAs) contain the same information as a MIB, plus additional code that enables the IA to get the information needed to respond to an SNMP Manager's request, from the internals of the network device.

An IA is created by a process of development referred to as **Instrumentation**. An IA is created from a MIB or multiple MIBs. An IA can be created for every MIB. One IA can be created for a number of unique MIBs. This process consists of the creation of an IA, MIB or multiple MIBs (data) and the code for accessing and manipulating this data. See Instrumenting Objects for details on IA development.

MIBs and IAs are two separate entities. The IAs and MIBs are intricately related. A MIB resides as a document on the remote client, while IAs are internal to a network device. However, a MIB is the foundation in the creation of an IA. The MIB provides information to the network manager on the remote client, and the IA provides information to the SNMP agent on the network device.

Parent Topic:

SNMP Basic Concepts

Instrumentation Functions

Instrumentation functions are used to register an object and its MIB, and provide the functions NWSNMP uses when it receives Get and Set requests against an object.

Standard Object-Handling Macros

The following macros are used to define Object Control Blocks for objects instrumented with standard object-handling functions.

SAI_DEFINE_GROUP	Defines a group table-oriented object.
SAI_DEFINE_POINTER	Defines a pointer table-oriented object.
SAI_DEFINE_TABLE	Defines a simple table-oriented object.

Registration Functions

The following functions are used to define, register, and deregister MIBs:

SAIRegisterMIB	Registers a MIB definition with NWSNMP.
SAIDeregisterMIB	Deregisters a MIB definition that has been registered with NWSNMP.

Specific Object-Handling Functions

The specific object-handling instrumentation functions give the structure of calls your NLM™ must provide for each abstract object you instrument with your own functions instead of with the standard object-handling functions. NWSNMP calls the functions you provide to fulfill Get, GetNext, and Set requests. Athough based on the specific object-handling functions, your functions should have different names, expecially if you are instrumenting multiple objects.

In addition, the **OCBLookupInst** function defines the lookup function for group objects registered with the standard object-handling functions.

The specific object-handling functions are as follows:

OCBAbortSet	Aborts a previously checked set.
OCBCheckSet	Checks the validity of parameters for a Set request.
OCBCommitSet	Sets a list of attributes in an object.
OCBGet	Reads attributes from an object.
OCBGetNext	Reads attributes from the next object instance in an ordered set of objects.
OCBLookupInst	Looks up an instance within a group of arrays using NWSNMP standard object-handling functions.

Parent Topic:

NWSNMP Functions and Structures

Related Topics:

Instrumentation Structures

Instrumentation Object Structures

The following structures are used to define the attributes of instrumented objects:

AttrInfo	Attribute Information Block
AttrValue	Attribute Value Block
ObjControl	Object Control Block
ObjInfo	Object Information Block

Parent Topic:

NWSNMP Functions and Structures

Related Topics:

Instrumentation Functions

Instrumentation Using Specific Object-Handling

Functions

If you cannot organize your abstract object data into a simple table, pointer table, or group table, you must provide your own object-handling functions. You can look in the header (.H) file produced by the MIB compiler to determine which structures the MIB definition requires from yourNLM™ code. Each group object requires an Object Control Block. The Object Control Block provides pointers to the NWSNMP functions which handles requests for Get, GetNext, or Set requests.

Each block includes pointers to functions for manipulating an object, as well as context information for the functions. The Object Control Block is described in Instrumentation Structures.

NWSNMP invokes the appropriate functions to handle the various requests that can be performed on your object.

The following type definitions for object-handling functions give the structure of functions your NLM must provide for each abstract object you register. NWSNMP calls these functions to fulfill Get and Set requests.

Table . Specific Object-Handling Functions: Instrumentation Prototype Definitions

Function	Purpose
OCBAbortSet	Aborts a previously checked set.
OCBCheckSet	Checks the validity of parameters for a Set request.
OCBCommitSet	Sets a list of attributes in an object.
OCBGet	Reads attributes from a specific object.
OCBGetNext	Reads attributes from the next object instance in an ordered set of objects.
OCBLookupInst	Looks up an instance within a group of arrays using the NWSNMP standard access methods.

The functions are all quite similar in nature, so some common points are covered here. These common points are process context, instance list handling, attribute list handling, and error handling. Detailed information about each function is provided in NWSNMP Functions and Structures.

Process Context

NWSNMP invokes your NLM object-handling functions on the NWSNMP thread. Thus, NWSNMP can invoke an object-handling function in any process context. For example, the context can be in a NetWare C Library thread or a NetWare operating system process, and in either case the context is not necessarily related in any way to the object-handling NLM. Consequently, when writing object-handling functions, avoid C Library functions except the context-free functions such as the memory and string manipulation functions.

> **IMPORTANT:** Do not block in your object-handling functions unless it is absolutely necessary. In particular, **do not** block in the **OCBCommitSet** function.

If your NLM depends on its own C Library context (that is, it uses one or more C Library functions), put the process into its own C Libarry context by calling **SetThreadGroupID**. In addition, reset the thread group immediately before returning.

Instance Lists

NWSNMP passes to each abstract object-handling function an array of Attribute Value Blocks that contain instance information. For abstract objects with only a single instance, such as statistics tables, this list is always empty and can be ignored. For tabular objects, this list contains the group of attributes that describe the instance involved in the request.

This instance list **always** contains blocks for all the variables in the INDEX clause of the abstract object in the same order in which they appear in the INDEX clause. (When NetWare SNMP receives a GetNext request with insufficient instance information, it completes the instance list with NULL values.)

Attribute Lists

NWSNMP passes an attribute list to each object-handling function when it calls the function. This list is an array of Attribute Value Blocks. These blocks contain the attributes that are involved in the current request. The abstract object code must traverse the list of attributes and perform a Get or Set request on the indicated value for each. The object can look at *avb_info->aib_id* to determine which attribute is being addressed.

NWSNMP does not order attribute lists; it passes the attributes to the object handler in the same order in which they appear in the original SNMP packet. In addition, NWSNMP does not discard duplicates, so an attribute list might contain multiple references to the same attribute. Your handling function must process these duplicate blocks.

Error Handling

Your object-handling functions must return SNMP_ERR_NOERROR when successful or one of the predefined error codes when they fail, indicating the reason the request cannot be completed. The following table lists NWSNMP return codes, which are defined in the SNMPAGNT.H header file.

Table . Object-Handling Routing Return Values

Return Value	Returned By	Description
SNMP_ERR_NOERROR	All	No error; the function succeeded.

SNMP_ERR_TOOBIG	**OCBGet, OCBGetNext**	The response exceeds the space allocated to it.
SNMP_ERR_NOSUCHNAME	All	The indicated item does not exist. This must be returned if the instance ID indicates an instance that does not exist or if the object handler encounters an unrecognized attribute. In the latter case, the Attribute Value Block must also be flagged.
SNMP_ERR_BADVALUE	**OCBCheckSet**	An **OCBCheckSet** function returns this if it detects an attempt to set an attribute to an invalid value.
SNMP_ERR_GENERR	All	General error not covered by any of the other error codes. For example, a memory allocation failure.

If the reason for the failure is specific to a particular attribute in the attribute list, your function must take the following steps when returning the error code:

1. Set to NULL the avb_info field of AttrValue block for the attribute that causes the failure.

2. Write the error code into the block's avb_length field.

Performing the above steps help identify the attribute that caused the failure to NWSNMP.

If your function returns an error code but does not flag a particular attribute, NWSNMP assumes that the failure applies generally to all the attributes in the request.

For example, if a Get request asks for information from a nonexistent object instance, the **OCBGet** function simply returns SNMP_ERR_NOSUCHNAME without flagging any particular attribute. In contrast, if the object exists but one of the queried attributes does not, the **OCBGet** function returns the SNMP_ERR_NOSUCHNAME error in the avb_length field for that attribute.

Parent Topic:

Specific Object-Handling Functions

Related Topics:

Instrumentation Using Standard Object-Handling Functions

Instrumentation Using Standard Object-Handling Functions

The NWSNMP instrumentation interface provides standard functions for three data organizations: simple table, pointer table, and group table. The SNMPGEN.H header file contains all the definitions for using the standard access functions. You must include this file in your code if you are using standard functions for any object you instrument.

When you instrument with standard object-handling functions, you must provide a macro call to NWSNMP to set up each object. The macro call creates an Object Control Block for the object and places the appropriate object-handling functions in the block. The Object Control Block is described in Instrumentation Structures.

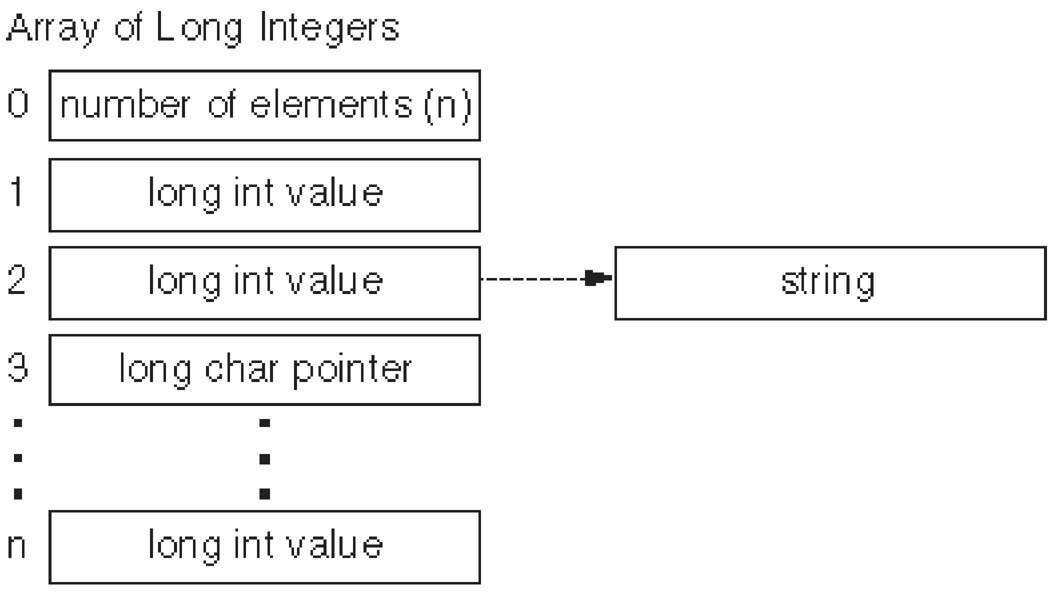
Simple Table

A simple table is an organized array of unsigned long values indexed by an attribute number. Immediate values are stored directly in the array fields. For pointer values, a pointer to the value is stored in the array. Array index zero contains a value giving the number of elements in the array. The following figure shows a simple table.

For example, simple tables can be used for statistics tables where most of the values are unsigned long data types.

Use the SAI_DEFINE_TABLE macro to define the Object Control Block for a simple table.

Figure 10. Simple Table

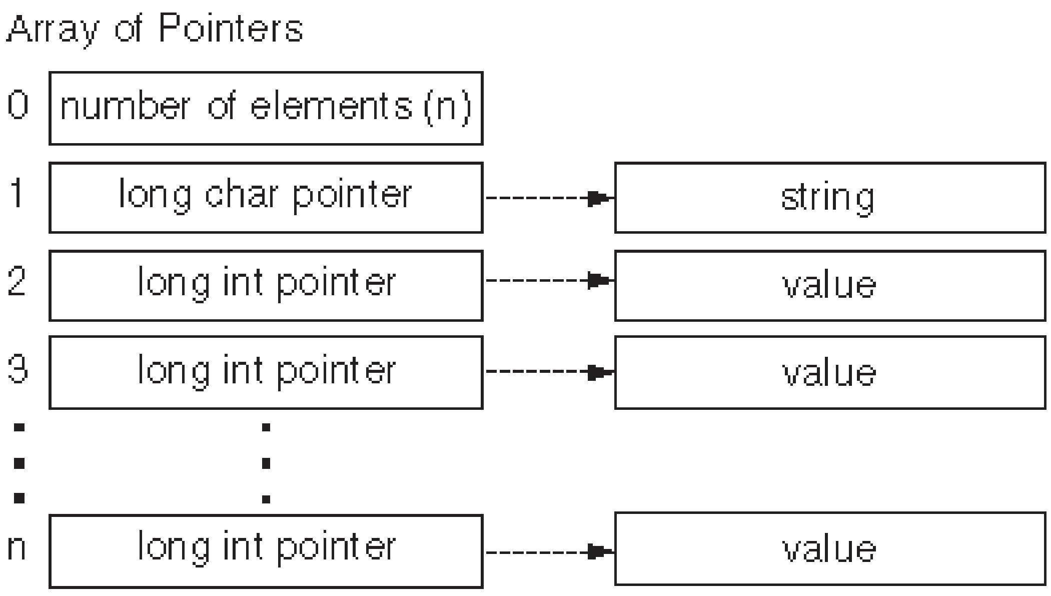


Pointer Table

The pointer table (see the following figure) is like the simple table, except it contains immediate values, which are values that exist directly in the simple table array. This allows for the simple instrumentation of values that cannot themselves be reorganized into a standard format. The pointers in the array are handled identically as in the simple table.

Use the SAI_DEFINE_POINTER macro to define the Object Control Block for a pointer table.

Figure 11. Pointer Table



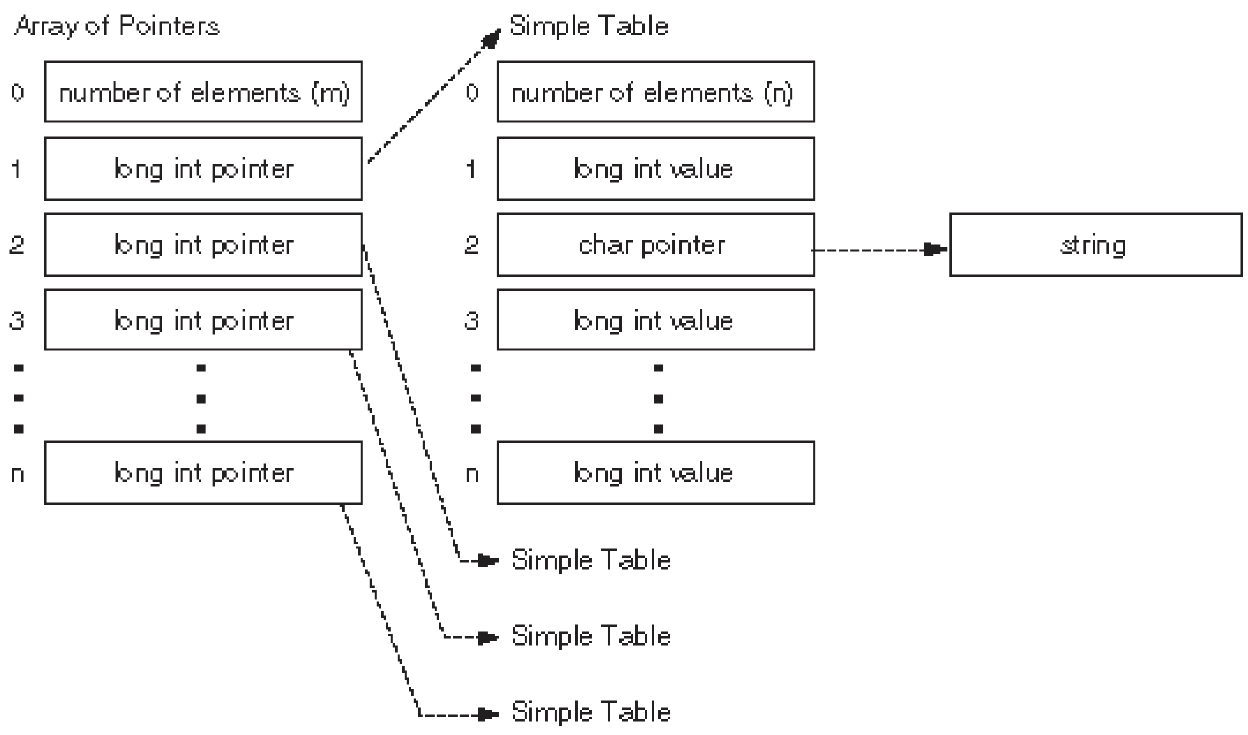
Group Table

A group table is a two-dimensional structure, an array of simple table arrays (see the following figure). Use a group table to instrument objects with multiple instances.

The organization of the attribute information is identical to the simple organization, but NWSNMP looks up the location of the array containing an object before manipulating the object. The **OCBLookupInst** function handles this step and returns a pointer to the location of the particular object.

Use the **SAI_DEFINE_GROUP** macro to define the Object Control Block for a group table. In addition, you must provide a function based on the type definition **OCBLookupInst** for the group table.

Figure 12. Group Table



Defining Values for a Standard Table

In all tables formatted for use with the standard access methods, the following values must be NULL-terminated:

Non-immediate values in a simple table

Any OCTET STRING value in a pointer table

In addition, writable non-immediate values must be stored in buffers large enough to hold the maximum length value. The length of attributes with no maximum is arbitrarily capped at 256 bytes, including the terminating null.

Example

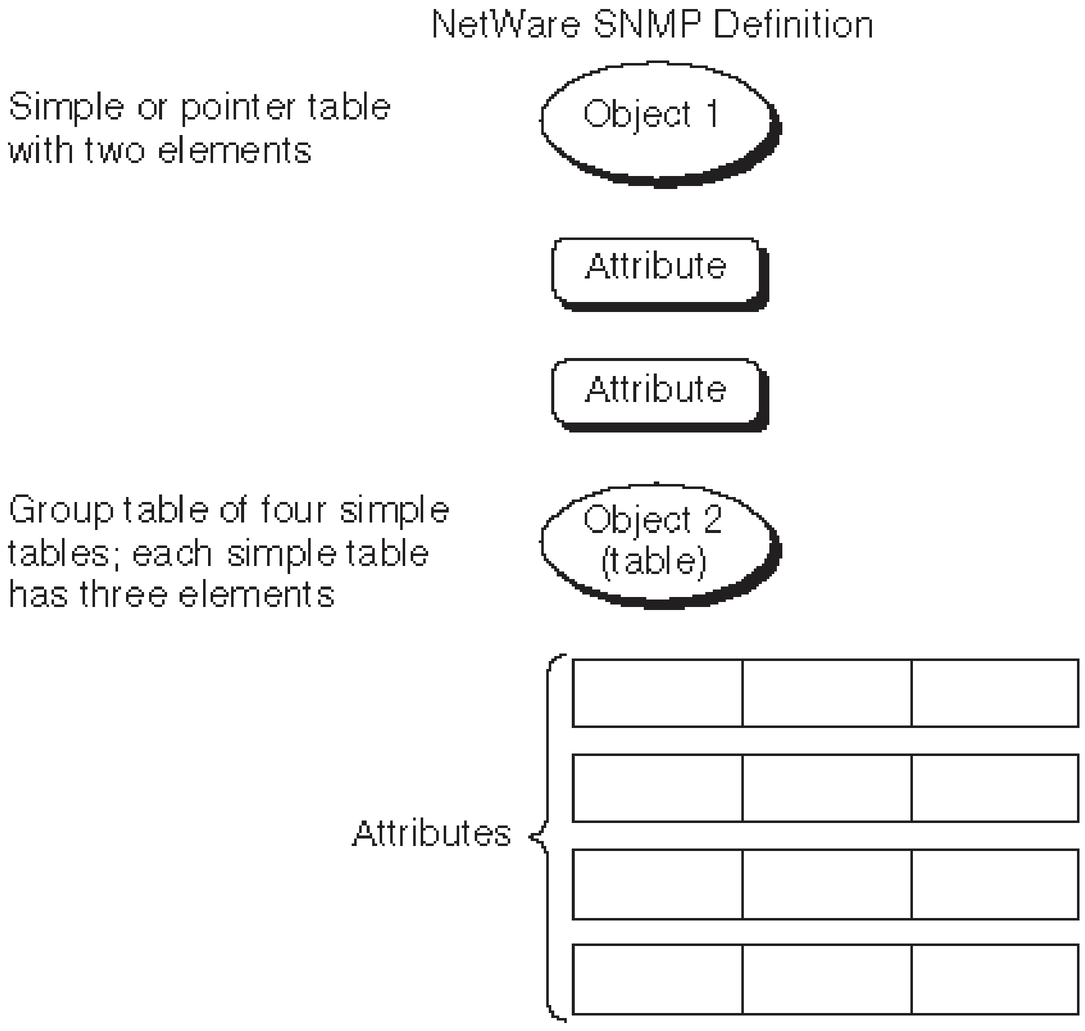
As an example, consider the instrumented object shown in the figure below. The MIB compiler interprets the MIB as two abstract objects. One has two attributes; the second is a series of arrays. Using this interpretation, the objects can be implemented as follows:

Object 1 can be represented as a simple table with two elements representing the attributes. If the variable values cannot fit in four bytes, the table can be implemented as a pointer table instead.

Object 2 can be represented as a group object of four elements, where each element is a simple table with three elements.

The following figure diagrams the instrumented object structure.

Figure 13. Instrumenting an Object with Standard Object-Handling Functions


NetWare SNMP Definition
Simple or pointer table with two elements
Object 1
Attribute
Attribute
Group table of four simple tables; each simple table has three elements
Object 2 (table)
Attributes

Parent Topic:

Standard Object-Handling Functions

Related Topics:

Instrumentation Using Specific Object-Handling Functions

Manageable Object Definitions

Object definitions can be scalar or they can be presented in tables.

All Objects

All SNMP manageable objects are defined using the OBJECT-TYPE macro, which performs a value assignment to define the object identifier for the object and also captures the semantics of the object.

The Sample Object Definition example shows an example of a typical object definition.

Sample Object Definition

```
 tcpRtoAlgorithm OBJECT-TYPE
SYNTAX              INTEGER {
             other(1),       -- none of the following
             constant(2),    -- a constant rto
             rsre(3),        -- MIL-STD-1778, Appendix B
             vanj(4)         -- Van Jacobson's algorithm
      }
 ACCESS    read-only
 STATUS    mandatory
 DESCRIPTION
     "The algorithm used to determine the time-out value used for
     retransmitting unacknowledged octets."
   ::= { tcp 1  }
```

The following paragraphs explain all the clauses in the new manageable object defined in the Sample Object Definition example.

`tcpRtoAlgorithm` is the textual name for the object. Its object identifier (or "**on the wire**" name) is tcp (1) or 1.3.6.1.2.1.6.1 (because tcp was defined, in Global Naming Tree example in The Global Naming Tree and ASN.1 Notation, to have the value 1.3.6.1.2.1.6).

SYNTAX specifies the type of the object, which, in this case, is INTEGER. In fact, it is an "enumerated INTEGER." The value of an enumerated INTEGER is restricted to one of the explicitly listed values, which, in this case, are 1, 2, 3, or 4. The name associated with each value is for convenience only. On the wire, the value is impossible to differentiate from any other INTEGER value.

The SYNTAX clause can specify any of the legal SNMP types described in the table entitled Values for Scalar Objects. You can also see "subtypes" of these types. A subtype further restricts the set of values a type can take.

For INTEGER types, a subtype takes the form of an inclusive range of values. In the following example, the integer is restricted to values in the range 0 to 65535 inclusive.

```
INTEGER(0..65535)
```

For OCTET STRING types, a subtype can be defined that has either a fixed length or a range of possible lengths. For example:

```
OCTET STRING (SIZE(6)); OCTET STRING (SIZE (0..255)); DisplayString (
```

ACCESS specifies the maximum access for the object. Your access rights can be less than the specified maximum value of 28. Access values depend upon the community string used and the particular implementation of the MIB. Possible access values include: read-only, read-write, write-only, and not-accessible.

These are all self-explanatory, except for not-accessible. Tables and rows within tables cannot be directly accessed using SNMP. Only individual table entries are accessible. Hence, table and row level definitions are flagged not-accessible. Objects that are defined to be used purely as parameters of traps are also often defined not-accessible, because you cannot access them directly.

STATUS provides information about the state of the MIB object. The following are possible values:

Mandatory---indicates the object must be implemented.

Optional---is never used. The SNMP convention is that if any objects within a group are implemented, then all objects within the group must be implemented. Hence, the unit of compliance is the group, not an individual object. A group is a collection of individual objects under a common node.

Obsolete---indicates the object is no longer supported.

Deprecated---indicates the object is being phased out. At some time in the future, it might become obsolete.

DESCRIPTION provides a textual description of the semantics of the object.

REFERENCE (not shown in this example) has the same form as the DESCRIPTION clause. It is sometimes used to provide a textual reference to a definition in some other document, such as an IEEE or ISO standard.

Table Objects

In MIBs, table definitions usually comprise four pieces:

The table object itself is defined with the SYNTAX "SEQUENCE OF <TableEntry>".

The row object is defined under the table node with the SYNTAX "<TableEntry>".

The type assignment for <TableEntry> defines the fields within the table.

The object definitions for each field are hung under the entry node.

The Sample Table Definition example provides a definition of tcpConnTable used in Global Naming Tree example in The Global Naming Tree and ASN.1 Notation.

Sample Table Definition

```
INDEX  { tcpConnLocalAddress,
                tcpConnLocalPort,
                tcpConnRemAddress,
                tcpConnRemPort }
        ::= { tcpConnTable 1 }

TcpConnEntry ::= SEQUENCE {
      tcpConnState         INTEGER,
      tcpConnLocalAddress  IpAddress,
      tcpConnLocalPort     INTEGER (0..65535),
      tcpConnRemAddress    IpAddress,
      tcpConnRemPort       INTEGER (0..65535)
   }

   tcpConnState OBJECT-TYPE
   SYNTAX     INTEGER
         .
         .
   ::= { tcpConnEntry 1 }

   tcpConnLocalPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
        .
        .
   ::= { tcpConnEntry 3 }

   tcpConnRemAddress OBJECT-TYPE
    SYNTAX IpAddress
        .
        .
   ::= { tcpConnEntry 4 }

   tcpConnRemPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
        .
        .
   ::= { tcpConnEntry 5 }
```

Most of the clauses in this example have been explained in Manageable Object Definitions. The following paragraphs explain the INDEX and DEFVAL clauses:

INDEX, which appears in the table entry definition, defines how to name an instance of an object within the table. For instance, in tcpConnTable, the index required to uniquely identify a row is formed from the ordered set of values: tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress, tcpConnRemPort. (See Table Objects for more information.)

DEFVAL (not shown in this example) is sometimes used within the OBJECT-TYPE macro, defining objects that are part of a table. This clause defines an acceptable default value for the object. Examples of DEFVAL clauses are shown below:

DEFVAL Clauses

```
DEFVAL { 0 }                  -- INTEGER
DEFVAL { `00000000'h }        -- hexadecimal string
DEFVAL { {0 0} }                -- NULL OBJECT IDENTIFIER 0.0
```

Parent Topic:

MIB Module Contents

Related Topics:

Trap Definitions

Value Assignments

Type Assignments and Textual Conventions

MIBs

To instrument an object, you must acquire or develop a MIB that describes that object (see Obtaining RFCs and MIBs). MIB development is discussed in Working with MIBs. The information from the MIB is then included in your NLM™, and NWSNMP uses the MIB information to read data and respond to SNMP requests regarding your instrumented objects.

Parent Topic:

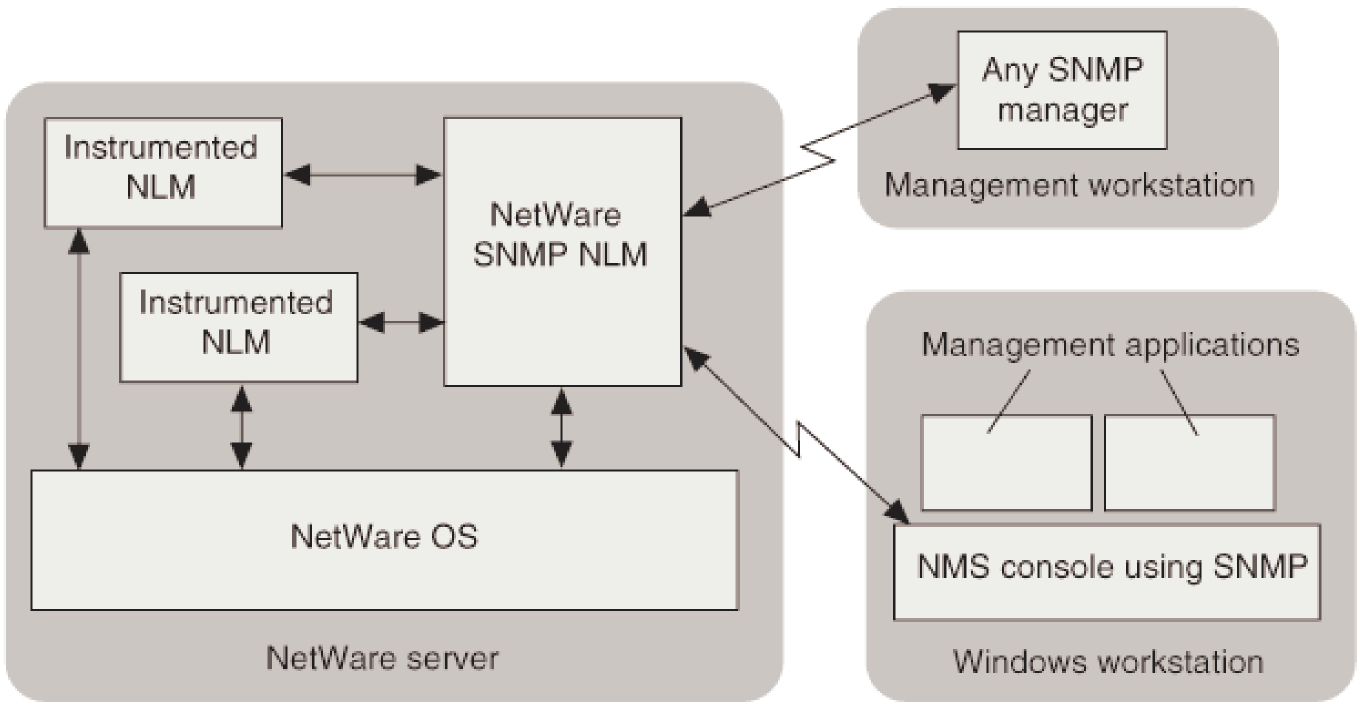
NWSNMP Concepts

NWSNMP Architecture

The following figure shows how NWSNMP and NLM™ software has been instrumented to interact with the NetWare operating system and management applications. An SNMP Manager can send SNMP requests to

NWSNMP, which acts as an interface between the SNMP manager and instrumented NLM files. An NLM that has been instrumented to NWSNMP is manageable by any SNMP remote client. For example, NWSNMP acts as an interface between the NMS Console and instrumented NLM files. Thus, an object instrumented to NWSNMP is manageable by applications compatible with the NMS console.

Figure 14. NWSNMP and NLM Software Architecture



Parent Topic:

NWSNMP Concepts

NWSNMP Get Handling

The Get and GetNext requests are very similar. As NWSNMP processes a Get or GetNext request, it performs the following actions:

1. NWSNMP decodes the SNMP request and prepares to get the requested information. In the process, it completes the following steps:

   a. NWSNMP identifies the authority for the request (by way of the community name and any other authorization features implemented) and the SNMP objects being requested.

b. NWSNMP looks up each SNMP object requested in the data store of registered abstract objects. From the abstract object information, NWSNMP identifies the abstract object ID, the attribute ID, and the instance ID. If the request does not have the authority to read this object, NWSNMP rejects the request.

c. If the request does have the proper authority, NWSNMP groups all requests for each object ID/Instance ID pair so they can be treated as a single request for information about the indicated object instance.

d. For each object instance, NWSNMP gets the data type of the object data.

e. NWSNMP converts the instance ID into native format. The request fails if the index is invalid: too long, too short, out of range, and so on.

2. If the decode process has been successful, NWSNMP calls the appropriate **OCBGet** or **OCBGetNext** function for the object, passing the Object Information Block, the instance ID, and the list of attributes to be read.

3. NWSNMP receives the requested information form the instrumented object.

4. Using the information returned by **OCBGet** or **OCBGetNext**, NWSNMP builds and sends an SNMP reply message, converting each piece of object information from native format into the appropriate ASN.1 format.

Parent Topic:

NWSNMP Client Interface

Related Topics:

NWSNMP Set Handling

Client Interface Return Values

Get and GetNext Handling

Set Handling

NWSNMP Set Handling

NWSNMP processes SNMP Set requests one at a time, using a two-phased commit to ensure that each Set is processed as a unit even if it affects multiple objects. If, when NWSNMP tries to process a Set request, it finds that it is currently blocked while processing a previous Set, it blocks the new Set until the previous Set is completed. Once it is ready to process a particular Set request, NWSNMP takes the following steps:

1. NWSNMP decodes the SNMP request and prepares to complete the Set. In the process, it completes the following steps:

    a. NWSNMP identifies the authority for the request (by way of a community name and any other authorization features implemented) and the SNMP objects being requested.

    b. NWSNMP looks up each SNMP object requested in the data store of registered abstract objects. From the abstract object information, NWSNMP identifies the abstract object ID, the attribute ID, and the instance ID. The request fails if the object is not writable or if the request does not have the authority to write this object.

    c. NWSNMP groups all requests for each object ID/instance ID pair so they can be treated as a single request to modify the indicated object instance.

    d. NWSNMP converts all attribute values from ASN.1 format into native format.

    e. For each object instance, NWSNMP gets the type of the object index from the object data store entry. NWSNMP converts the instance ID into native format. The request fails if the index to any of the requested objects is invalid: too long, too short, out of range, and so on.

2. If the decode process has been successful, NWSNMP calls the appropriate **OCBCheckSet** function for each object, passing the object ID, the instance ID, and the list of attribute values to be set. If **OCBCheckSet** fails for any object instances, NWSNMP aborts the request by calling the appropriate **OCBAbortSet** function for all object instances that previously responded successfully to **OCBCheckSet**.

3. The instrumented NLM™ sends a response to NWSNMP. The response indicates whether the object can complete the Set.

4. If all calls to **OCBCheckSet** are successful for the request, NWSNMP calls the **OCBCommitSet** function for each object instance, passing the object ID, the instance ID, and the list of attribute values to be set.

    IMPORTANT: If **OCBCheckSet** succeeds, **OCBCommitSet** should also succeed.

5. The instrumented NLM sends a response to NWSNMP indicating the Set has been completed.

6. NWSNMP builds and sends an SNMP reply message indicating whether the Set succeeded.

Parent Topic:

NWSNMP Client Interface

Related Topics:

Client Interface Return Values

NWSNMP Get Handling

Get and GetNext Handling

Set Handling

Object Control Blocks

For each abstract object, the MIB compiler produces C code that refers to an external Object Control Block of type struct ObjControl. This block must be defined by the object-handling code. The name of the control block is always the name of the abstract object with the prefix "OCB_." For example, the IP statistics object is described by OCB_ip, and the IP routing entry object is described by OCB_ipRouteEntry.

> **NOTE:** The extern statements in the header (.H) file that the MIB compiler generates name the Object Control Blocks you must define.

Each node of the MIB tree with accessible subbranches is an abstract object and requires an Object Control Block. Nodes without accessible subbranches are not abstract objects. For example, ipRouteTable does not have a control block because its only subbranch is ipRouteEntry, which has an ACCESS value that is not-accessible. The ipRouteEntry node does have several accessible subbranches. Therefore, it does require a control block.

Parent Topic:

Instrumentation Structures

Related Topics:

Attribute Information Blocks

Attribute Value Blocks

Object Information Blocks

Object Information Blocks

An Object Information Block contains four pointers, as follows:

The *oib_id_pt* field points to an integer array. The array contains the ID of the object in the MIB, one sub-ID per array element.

The *oib_control* field points to the Object Control Block.

The *oib_attr_pt* field points to an array of Attribute Information Blocks. Each block in the array describes one of the attributes of the object. The array contains one block for each attribute.

The *oib_index_pt* field points to a second array of Attribute Information Blocks that serve only as indexes. These indexes are used to find attributes for objects that have multiple instances, such as tables. (The Sample Attribute Information Block figure shows an example.)

Parent Topic:

Instrumentation Structures

Related Topics:

Object Control Blocks

Attribute Information Blocks

Attribute Value Blocks

Object Registration and Deregistration

Registration allows NWSNMP to manage a dynamically changing set of objects, depending on which NLM™ files are loaded on the server. An NLM registers its instrumented objects at load time, and deregisters them when it unloads. Thus, NWSNMP always has current information about which objects it can manage.

The set of data associated with objects that have been registered--that is, the data NWSNMP can access--is the NWSNMP **data store**.

Parent Topic:

NWSNMP Concepts

Obtaining MIBs

The Internet Engineering Task Force (IETF), the standards body that created TCP/IP and SNMP, has created several standard MIBs for common network devices and protocols. These standard MIBs are published as RFCs and are available through anonymous FTP from host 192.33.33.22 and other sites. The standard MIBs are under the `{ ...internet(1) mgmt(2) }` node of the naming tree.

The RFC directory contains all published RFCs. The internet-drafts directory contains the latest working versions of documents that have not yet reached standards status.

Proprietary MIBs are available under each vendor's assigned node in the `{...internet(1) private(4) enterprises(1) }` branch of the naming tree. For instance, all Novell proprietary MIBs are under `{...internet(1) private(4) enterprises(1) novell(23) }`.

Often vendor proprietary MIBs are posted on host 128.9.0.32 and can be obtained through anonymous FTP as well.

Parent Topic:

Management Information Bases

Related Topics:

The Two Uses of MIBs

The Global Naming Tree and ASN.1 Notation

Three Types of SNMP Objects

MIB Module Contents

Obtaining RFCs and MIBs

RFCs

RFCs are documents used to propose Internet (TCP/IP) protocols and procedures. They provide the specification for all Internet Protocols. The RFCs are available at the following World Wide Web address:

```
http://www.internic.net/ds/dspg0intdoc.html
```

In addition, you can use electronic mail (e-mail) to request RFCs from the InterNIC Directory and Database Services automated mail server. To do this, send a message to `mailserv@ds.internic.net`. In the body of the message, indicate the RFC you want, as follows, where **NNNN** is the number of the RFC:

```
document-by-name rfcNNNN
```

For RFCs in the PostScript format, specify the extension, as follows:

```
document-by-name rfcNNNN.ps
```

To send multiple requests in a single message, specify each request on a separate line.

Use the following line to request the index of RFCs:

```
document-by-name rfc-index
```

The following protocol specifications, as well as other protocol-related information, are gathered in the DDN Protocol Handbook, which is available from the following centers:

DDN Network Information Center
14200 Park Meadow Dr., Suite 200
Chantilly, VA 22021, USA

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314, USA

The following RFCs define and provide background information relating to SNMP:

RFC 1155, *Structure and Identification of Management Information for TCP/IP-based Internets*, Rose, M. and McCloghrie, K., May 1990.

RFC 1157, *A Simple Network Management Protocol*, Case, J. and others, May 1990.

RFC 1212, *Concise MIB Definitions*, Rose, M. and McCloghrie, K., March 1991.

RFC 1213, *Management Information Base for Network Management of TCP/IP-Based Internets: MIB II*, McCloghrie, K. and Rose, M., March 1991.

RFC 1215, A *Convention for Defining Traps for use with the SNMP*, Rose, M., March 1991.

MIBs

To obtain published MIBs using anonymous ftp, log in to **venera.isi.edu** (IP address **128.9.0.32**) with the login name **anonymous**. Give your E-mail address as the password. Public MIBs are located in the **mib** directory.

Parent Topic:

NWSNMP Introduction

Related Topics:

Management Information Bases

OCTET STRING Conversion

Short OCTET STRING types are intended as a convenient way of handling OCTET STRING types that fit in the four-byte length field of the Attribute Value Block. Handling these as a special case eliminates the need for a buffer, in addition to the Attribute Value Block, to pass these values around.

Storing the strings as unsigned long values does not change the characteristics of the OCTET STRING types. An IpAddress is still an array of four bytes and it continues to exist as an array of four bytes even when it resides in an unsigned long field. When running on a machine that uses Intel* format, this means that the first byte of the OCTET STRING is in the low order byte of the unsigned long.

Therefore, when ordering short OCTET STRING types lexicographically, you **must** perform a byte-by-byte comparison. Comparing the actual unsigned long values for greater than or less than does not deliver the desired results.

Parent Topic:

Universal Tasks Handled by NWSNMP

Related Topics:

Grouping

Data Conversion

Time Field Conversion

Abstract Objects with Exactly One Instance

Attribute Range Checking

Operational Flow of an Instrumented NLM

The operational flow of an instrumented NLM™ is as follows:

1. Register with NWSNMP when loading the NLM.
2. Continue ordinary request. NWSNMP calls object management functions when needed.
3. Deregister with NWSNMP when unloading.

Parent Topic:

NWSNMP Concepts

Request/Response Transaction

In a **request/response transaction**, the network manager sends a request packet, referred to as a Protocol Data Unit (PDU), to the SNMP agent in a network device. The SNMP agent searches for the response, and then sends back a response PDU to the network manager.

There are three types of requests:

Get, which allows the network manager to ask the SNMP agent for the current value of one or more objects, in random order, in the network device. The SNMP agent gathers the requested data and sends it back to the manager.

GetNext, which is essentially the same as Get, except that it allows a manager to retrieve data from a table in sequential order.

Set, which allows the network manager to ask the SNMP agent to change the value of one or more objects to specified values. The SNMP agent performs the Set and sends a confirmation back to the network manager. If the network manager requests multiple objects to be set in the same PDU, the SNMP agent is mandated to set them all **as if simultaneously**. In particular, the SNMP agent must set either all or none of the objects.

For each request PDU, there is one response PDU. In the case of a Get or GetNext request, the response PDU contains either the requested data or an error code. For a Set request, the response PDU returns with an error status if the Set failed. If the Set did not fail, the response is identical to the request and all variables are updated.

In summary, the request/response transaction is performed as follows:

1. The network manager makes a management request by sending a request PDU to the SNMP agent on the network device.
2. The SNMP agent receives the request.
3. The SNMP agent decides which IA can handle the request.
4. The SNMP agent calls the appropriate IA's Get, GetNext, or Set request to get the response.
5. The response is sent back from the IA in the form of a response PDU to the SNMP agent and returned to the network manager on the remote client.
6. The network manager receives the response and the management task is verified and completed.

Parent Topic:

SNMP Operation

Related Topics:

Trap Transaction

Get, GetNext, and Set Functions

Required Software

To develop an NLM using the NWSNMP, you must have a network compiler SDK that allows you to develop NLM™ files for the NetWare platform. This can be found in the NetWare SDK (see NLM Applications). Certain C library calls must be used in association with the NWSNMP API functions to register with NWSNMP successfully.

To run the MIB compiler, you must have MS-DOS* 3.3, MS-DOS 6.0, or DR DOS 6.0 software or Novell DOS™ 7.0 software. The MIB Compiler has not been tested on other versions of DOS.

To obtain Novell products, call 1-800-NETWARE (1-800-638-9273) or

1-801-861-5588 in the U.S. or Canada, or contact your local Novell reseller.

To obtain SDKs, or for additional information on development tools and developer support programs, call 1-800-NETWARE (1-800-638-9273) or 1-801-861-5588 in the U.S. or Canada, or contact your international Novell office.

Parent Topic:

NWSNMP Introduction

Scalar Objects

Scalar objects represent an individual piece of management information. The following table lists and explains the scalar object values.

Table . Values for Scalar Objects

Value	Explanation
INTEGER	A signed whole number. Although SNMP imposes no size limit, most implementations restrict an INTEGER to 32 bits.
Counter	A non-negative integer that monotonically increases until it reaches a maximum value (232-1), when it wraps around and starts increasing again from zero. As the name implies, Counters are typically used to count notable events in the system. The absolute value of a Counter is often less useful than its delta value. Rate information is derived from the last sample.
Gauge	A non-negative integer that can increase or decrease, but which can never exceed the absolute maximum value (232-1). The value of a Gauge is at maximum whenever the information being modeled is greater than or equal to that maximum. If the information being modeled subsequently decreases below the maximum value, the Gauge also decreases.
TimeTicks	A non-negative integer that represents the time (modulo 232) in hundredths of a second since some epoch. For example, an object of type TimeTicks can be used to model how long a system has been up and running since it was started.
IpAddress	A 32-bit Internet address.
OCTET STRING	A sequence of octets (bytes). Use an OCTET STRING to represent the following: (1) A string of printable characters, such as the

	name of a system. (2) Any arbitrary binary data, such as a Macintosh address.
OBJECT IDENTIFIER	A number scheme, ASN.1 Notation, that represents the name of a manageable object. See Trap Transaction for more information about this value.
Other Types	SNMP allows three other types: NULL, Opaque, and NetworkAddress. However, these are never used and can safely be ignored.

Parent Topic:

Three Types of SNMP Objects

Related Topics:

Table Objects

Group Objects

Set Handling

The SNMP agent processes Set requests one at a time, using a two-phased commit to ensure that each Set is processed as a unit even if it affects multiple objects. When the SNMP agent tries to process a Set request, if it finds that it is currently blocked while processing a previous Set, it blocks the new Set until the previous Set is completed. Once it is ready to process a particular Set request, the SNMP agent takes the following steps:

1. The SNMP agent decodes the SNMP request and prepares to complete the Set. In the process, it completes the following steps:

    a. The SNMP agent identifies the authority for the request (by way of a community name and any other authorization features implemented) and the SNMP objects being requested.

    b. The SNMP agent looks up each SNMP object requested in the data store of registered abstract objects. From the abstract object information, the SNMP agent identifies the abstract object ID, the attribute ID, and the instance ID. The request fails if the object is not writable or if the request does not have the authority to write to this object.

    c. The SNMP agent groups all requests for each object ID/instance ID pair so they can be treated as a single request to modify the indicated object instance.

    d. The SNMP agent converts all attribute values from ASN.1 Notation format into the native format of the machine to which the ID arrives.

e. For each object instance, the SNMP agent gets the type of the object index from the object data store entry. The SNMP agent converts the instance ID into native format. The request fails if the index to any of the requested objects is invalid: too long, too short, out of range, and so on.

2. If the decode process has been successful, the SNMP agent calls the appropriate **OCBCheckSet** function for each object, passing the object ID, the instance ID, and the list of attribute values to be set. If **OCBCheckSet** fails for any object instances, the SNMP agent aborts the request by calling the appropriate **OCBAbortSet** function for all object instances that previously responded successfully to **OCBCheckSet**.

3. The IA sends a response to the SNMP agent. The response indicates if the object can complete the Set.

4. If all calls to **OCBCheckSet** are successful for the request, the SNMP agent calls the **OCBCommitSet** function for each object instance, passing the object ID, the instance ID, and the list of attribute values to be set.

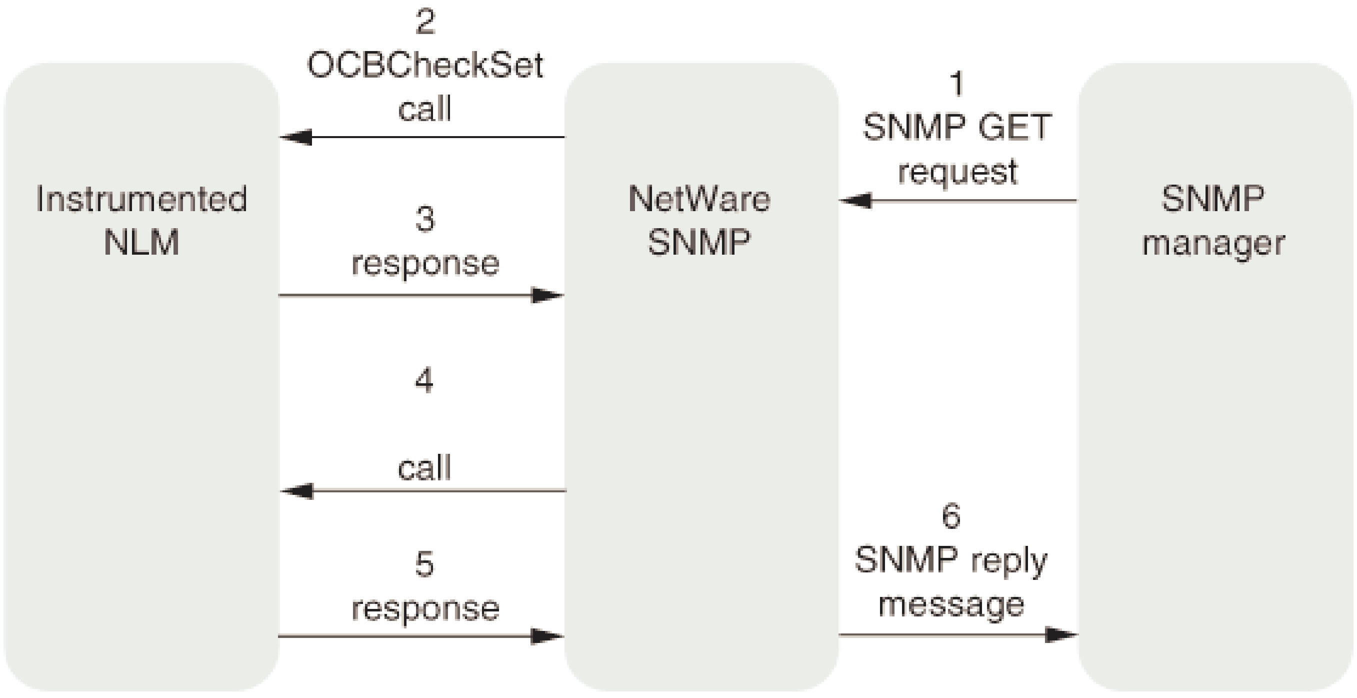
   NOTE: If **OCBCheckSet** succeeds, **OCBCommitSet** most likely will also succeed.

5. The IA sends a response to the SNMP agent indicating the Set has been completed.

6. The SNMP agent builds and sends a Protocol Data Unit (PDU) response message indicating whether the Set succeeded.

The following figure shows a Set request using the two-phased commit.

Figure 15. Set Handling Using the two-phased commit



Parent Topic:

Get, GetNext, and Set Functions

Related Topics:

Get and GetNext Handling

SNMP Introduction

SNMP is a protocol that offers network management services within the Internet (TCP/IP) suite of protocols. Most leading network products support SNMP, allowing vendor-independent management of network devices and applications.

SNMP is a datagram protocol that can run over several transport protocols, including IPX™ and UDP.

Benefits of SNMP include:

Minimal-impact design. SNMP is designed to be small, simple, and have minimal impact on the network device it is managing.

Datagram (connectionless, best-effort) protocol. The time you need network management the most is when the network is having trouble. Under these conditions, a connectionless protocol performs better than a connection-oriented protocol, which spends most of its time rebuilding dropped connections rather than moving data.

Network diagnostic ability. SNMP provides a polling feature that periodically polls the network device to see whether anything needs attention. If it finds that a request or response gets lost, it has the global knowledge to deal with the situation.

Trap-directed polling. When something goes wrong, such as the network device is running low on memory, the SNMP agent sends a message, called a **trap**, to the network manager, informing it of the status of a network device. The network manager can then intelligently poll the network device based upon the trap, to determine the exact problem.

Security. SNMP has a security mechanism whereby each packet contains a string called the **community string**. The community string acts as a combination of the user name and password and is used by a network device to determine the level of management control a network manager has over a network device.

Network management services. SNMP provides network management services to the network, including installation, maintenance, configuration, performance, and fault management.

If you are unfamiliar with SNMP management, you will need to understand the following definitions:

Management objects: An item of configuration that can be managed on the network.

Network devices: Includes a number of different SNMP-managed devices such as routers, gateways, and hosts.

Instrumentation: The process of writing **instrumentation code**. Instrumentation code resides on the network device and enables a network device to be managed on the network. With each implementation, instrumentation code is written according to development objectives. In most implementations, instrumentation code results in an Instrumentation Agent (see Instrumentation Agents).

Parent Topic:

SNMP Basic Concepts

Table Objects

SNMP allows data to be structured into tables. A table object is made up of rows and columns, similar to a table of information in a document. One way to think of a table is as a list of records where each record corresponds to a row in the table, and each field in the record corresponds to a cell, in which the value can be of any of the simple types described in Scalar Objects. Tables cannot be nested.

An example of a table is `tcpConnTable` (1.3.6.1.2.1.6.13) in the Global Naming Tree example (see The Global Naming Tree and ASN.1 Notation), which contains information about tcp. The following table, shows a fragment of tcpConnTable belonging to a host. The table is used in addressing the examples that follow.

Table . Sample SNMP Table Object

tcpConnState	tcpConnLocalAddress	tcpConnLocalPort	tcpConnRemAddress	tcpConnRemPort
2	0.0.0.0	9	0.0.0.0	0.0.0.0
2	0.0.0.0	19	0.0.0.0	0
2	0.0.0.0	23	0.0.0.0	0
2	0.0.0.0	111	0.0.0.0	0
6	130.57.4.252	1028	130.57.10.123	6000

SNMP does not allow requests on a table as a whole, or on rows within tables, because SNMP requests apply only to individual scalar objects. It handles table requests as individual table entries. A table entry is addressed by concatenating "instance" information to the end of the object identifier of the field you want to access. Thus, the first part of the resultant object identifier identifies the column within a row, and the latter part identifies the row.

Tables in SNMP are required to have an index that uniquely identifies each row in the table. This index can be either a single value or, if necessary to create a unique index, an ordered set of values. The index is defined by the designer of the table, and is usually (but not necessarily) the values of the fields within the table. For instance, in the previous table, in tcpConnTable, the index required to uniquely identify a row is formed from the ordered set of values---`tcpConnState`,`tcpConnLocalAddress`, `tcpConnLocalPort`,`tcpConnRemAddress`, and `tcpConnRemPort`.

The object identifier instance to access `tcpConnState` in the last row of the table shown in the previous table is as follows:

```
1.3.6.1.2.1.6.13.1.1.130.57.4.252.1028.130.57.10.123.6000
```

where:

1.3.6.1.2.1.6.13.1.1---is the object identifier of `tcpConnState`

130.57.4.252---specifies the local address of interest

1028---specifies the local port of interest

130.57.10.123---specifies the remote address of interest

6000---specifies the remote port of interest

You can access table entries in one of the following ways:

Randomly, by specifying the complete instance as described above and using the Get request.

Sequentially, using the GetNext request. When you supply a (possibly partial) object identifier instance with the GetNext request, it returns the value and object identifier instance of the next object in lexicographic order. If you subsequently do a GetNext on the returned object identifier instance, the next instance is returned, and so on. In this way, you can " **walk**" a table without any prior knowledge of the indexes.

For example, using the above table, GetNext 1.3.6.1.2.1.6.13.1.1 returns the following:

value---2; instance---1.3.6.1.2.1.6.13.1.1.0.0.0.0.9.0.0.0.0.0

GetNext 1.3.6.1.2.1.6.13.1.1.0.0.0.0.9.0.0.0.0.0 returns

value---2; instance---1.3.6.1.2.1.6.13.1.1.0.0.0.0.19.0.0.0.0.0

GetNext 1.3.6.1.2.1.6.13.1.1.0.0.0.0.19.0.0.0.0.0 returns

value---2; instance---1.3.6.1.2.1.6.13.1.1.0.0.0.0.23.0.0.0.0.0

GetNext 1.3.6.1.2.1.6.13.1.1.0.0.0.0.23.0.0.0.0.0 returns

value---2; instance---1.3.6.1.2.1.6.13.1.1.0.0.0.0.111.0.0.0.0.0

GetNext 1.3.6.1.2.1.6.13.1.1.0.0.0.0.111.0.0.0.0.0 returns

value---6;
instance---1.3.6.1.2.1.6.13.1.1.130.57.4.252.1028.130.57.10.123.6000

To add a new row to a table, a remote client usually sends a Set Protocol Data Unit (PDU) referring to a nonexistent object instance. The SNMP agent then creates the complete row containing that object instance. If the Set PDU does not contain a value for each field in the row, the SNMP agent normally provides a default value for that field. Some tables, notably those in the RMON MIB (RFC 1271), have a more complex row creation protocol, which is described in the appropriate Requests for Comment (RFC).

Parent Topic:

Three Types of SNMP Objects

Related Topics:

Group Objects

Scalar Objects

TCPIP and NWSNMP Load Information

NWSNMP allows command line parameters to change the community strings required for read requests, write requests, and trap generation. Each can be modified using the command line or INETCFG.

If NWSNMP is autoloaded by TCPIP, the default settings are used. By default, NWSNMP allows read operations to be performed using the community `public' string. Write operations are disallowed. To test write operations or to use another community string for read operations, NWSNMP must be loaded explicitly with the appropriate command line parameters for Monitor and Control. See INETCFG help for more information on configuring NWSNMP.

Parent Topic:

Loading NWSNMP

The Global Naming Tree and ASN.1 Notation

The **global naming tree** is an addressing scheme, central to MIBs, that points to objects contained in a MIB (for a definition of **objects**, see Three Types of SNMP Objects). This addressing scheme is a simple way of assigning a unique name to each SNMP object, which helps to identify each object during SNMP requests.

The global naming tree uses the hierarchical naming scheme, **Abstract Syntax Notation one** (ASN.1 Notation), developed by the International Standards Organization (ISO). In the ISO scheme, the structure is like a directory tree. The tree consists of a root connected to a multitude of labeled nodes or branches, descending downward. Each node might, in turn, have subnodes of its own (termed subordinates), which are also labeled. The figure below shows the global naming tree branches.

The root node of the global naming tree is unlabeled, and has three subnodes directly under it, including:

`ccitt(0)`---administered by the International Telegraph and Telephone Consultative Committee (CCITT), which issues recommendations that are adopted and recognized as standards internationally.

`iso(1)`---administered by the International Standards Organization (ISO), which is devoted to figuring standards for international data communications.

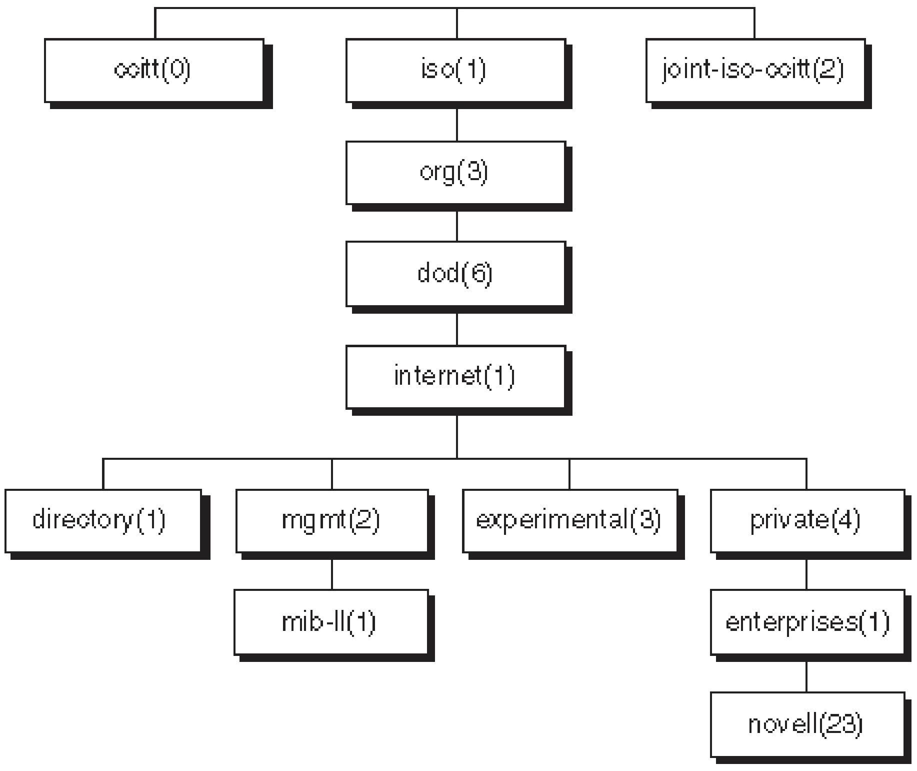
`joint-iso-ccitt`(2)---jointly administered by the ISO and the CCITT.

Each node has a label assigned to it. The label includes both a **textual description** and a **number**. An example of a label is iso(1). The textual description is iso. This part of the label is for convenience only. The number portion of the label (1) is called an **integer**. The figure below shows the location of the Novell's node---`Novell(23)`. The textual description is `Novell`; the integer is (23).

Each node has a specific location on the global naming tree. This location is called an **object identifier**. An object identifier is created by assigning a number to an object at a specific location on the global naming tree. As explained above, Novell has its own object identifier, which is `Novell(23)`. By beginning at the top of the global naming tree and descending downward to Novell's location, adding each node number, separated by a period, you would arrive at Novell's complete object identifier: 1.3.6.1.4.1.23.

Having been assigned an object identifier, Novell is free to assign names under its node without conflicting with names generated by other vendors.

Figure 16. Global Naming Tree Branches



The SNMP naming convention is very similar to a UNIX* or DOS fully qualified path name. The Global Naming Tree example shows part of a naming tree. All objects of interest to SNMP are in the iso(1) branch of the tree under the internet(1)node.

The Global Naming Tree

```
iso(1)
   org(3)
      dod(6)
         internet(1)
            directory(1)
            mgmt(2)
               mib-2(1)
                  system(1)
                  interfaces(2)
                  at(3)
                  ip(4)
                  icmp(5)
                  tcp(6)
                     tcpRtoAlgorithm(1)
                     tcpConnTable(13)
                        tcpConnEntry(1)
                           tcpConnState(1)
                           tcpConnLocalAddress(2)
                           tcpConnLocalPort(3)
                           tcpConnRemAddress(4)
                           tcpConnRemPort(5)
                     udp(7)
                     egp(8)
                     transmission(10
                     snmp(11)
                     rmon(16)
               experimental(3)
               private(4)
                  enterprises(1)
                     novell(23)
```

In the Global Naming Tree example, the `tcpRtoAlgorithm`, for example, is the textual descriptive name for the manageable object that indicates the tcp retransmission time-out algorithm in use on the managed network device. Its object identifier is 1.3.6.1.2.1.6.1.

Parent Topic:

Management Information Bases

Related Topics:

Three Types of SNMP Objects

The Two Uses of MIBs

MIB Module Contents

Obtaining MIBs

The Two Uses of MIBs

Management Information Bases are used on both the remote client and the network device. On the remote client, the network manager uses the MIB document to get the network device's management and configuration information. Every network manager handles the MIB document differently, depending on the design of the Network Manager.

On the network device, the SNMP agent uses the IA to get the network device's management and configuration information. A MIB is the base for IA development. For use on the network device, the MIB document is compiled by a MIB compiler to form .H header and .C source files. Additional code is added to the .H and .C files to create the IA. This process of developing an IA is known as **instrumentation**.

NOTE: Some SNMP documentation simplifies the two uses of MIBs by referring to the IA as a MIB on the network device side as well. In this documentation, a MIB resides on the remote client and an IA resides as a network device. Also, a MIB is only one part of the IA; it is not identical to an IA.

Parent Topic:

Management Information Bases

Related Topics:

The Global Naming Tree and ASN.1 Notation

Three Types of SNMP Objects

MIB Module Contents

Obtaining MIBs

Time Field Conversion

NWSNMP automatically converts attributes of type TimeTicks from internal NetWare operating system ticks to the SNMP 10 ms units. In most cases, TimeTicks objects contain the value of*sysUpTime* when a particular event occurred. To support such objects, yourNLM™ code should call **GetCurrentTime** and save the results to record the TimeTicks value. NWSNMP automatically converts the **GetCurrentTime** value to SNMP TimeTicks when the object attribute is accessed.

Parent Topic:

Universal Tasks Handled by NWSNMP

Related Topics:

Grouping

Data Conversion

OCTET STRING Conversion

Abstract Objects with Exactly One Instance

Attribute Range Checking

Trap Definitions

Trap definitions are very similar to object definitions, except that they use the TRAP-TYPE macro. The Sample Trap Definition example shows a sample trap definition.

Sample Trap Definition

```
linkDown TRAP-TYPE
ENTERPRISE snmp
VARIABLES  {ifIndex   }
DESCRIPTION
"A linkDown trap signifies that the sending protocol entity
 recognizes a failure in one of the communication links
 represented in the agent's configuration.
::= 2
```

The following paragraphs explain the clauses in the trap definition:

ENTERPRISE contains the object identifier of a node in the vendor's tree, which, along with the trap number (the 2 following the "::=" in the Sample Trap Definition example), uniquely identifies the trap.

VARIABLES defines an ordered sequence of MIB objects that are passed as parameters of the trap when it is generated.

DESCRIPTION provides a textual description of the semantics of the trap.

Parent Topic:

MIB Module Contents

Related Topics:

Value Assignments

Type Assignments and Textual Conventions

Manageable Object Definitions

Trap Interface Functions

The following function allows you to provide NWSNMP with the information it needs to forward a trap:

SACTrap	Generates an SNMP trap.

Parent Topic:

NWSNMP Functions and Structures

Related Topics:

Client and Trap Structures

Client Interface Functions

Trap Transaction

In a **trap transaction**, the SNMP agent sends an unsolicited notification contained in a trap Protocol Data Unit (PDU) to the network manager to inform it that some unexpected or extraordinary event has occurred. The SNMP agent might include the names and values of pertinent objects in the trap PDU to provide the network manager with additional information about the event. Note that there is no confirmation back from the network manager to the SNMP agent to say that it has received the trap.

In summary, the trap transaction is performed as follows:

1. An unexpected or extraordinary event occurs on the network device.
2. The SNMP agent sends a trap PDU to the network manager with a message to inform the manager of the problem.
3. The network manager is notified by the trap PDU event that the network device is in trouble. Depending upon the message, the network manager deals with the problem, but does not notify the SNMP agent that the message was received.

Parent Topic:

SNMP Operation

Related Topics:

Request/Response Transaction

Type Assignments and Textual Conventions

Type assignments give a name to a new type. In SNMP MIBs, this feature is mainly used to create what have become known as **textual conventions**. In textual conventions, a descriptive name is defined as an alias for some existing type, and additional semantics are defined in accompanying comments. This new descriptive type name is then used throughout the rest of the MIB to improve readability and clarity. Note that no new "on the wire" types are defined; this is merely a way to document the conventions on how to interpret a value of this type.

The Sample Textual Conventions example shows examples of textual conventions.

Sample Textual Conventions

```
DisplayString ::=
   OCTET STRING
-- This data type is used to model textual information taken
-- from the NVT ASCII character set. By convention, objects
-- with this syntax are declared as having SIZE (0..255)

PhysAddress ::=
   OCTET STRING
-- This data type is used to model media addresses. For many
-- types of media, this will be in a binary representation.
-- For example, an ethernet address would be represented as
-- a string of 6 octets.
```

Parent Topic:

MIB Module Contents

Related Topics:

Manageable Object Definitions

Trap Definitions

Value Assignments

Value Assignments

Value assignments are usually used to define object identifiers for interior nodes in the naming tree. The following are examples of value assignments:

`mib-2(1)` OBJECT IDENTIFIER::= { `mgmt 1` }

`mgmt` was imported from module RFC1155-SMI, where it is defined to have the value 1.3.6.1.2 (see Global Naming Tree example in The Global Naming Tree and ASN.1 Notation). Hence, identifier `mib-2`(1) of type object identifier is assigned the value 1.3.6.1.2.1.

`tcp` OBJECT IDENTIFIER ::= { `mib-2 6` }

`mib-2(1)` was previously defined to have the value 1.3.6.1.2.1. Hence, identifier tcp of type object identifier is assigned the value 1.3.6.1.2.1.6.

Parent Topic:

MIB Module Contents

Related Topics:

Type Assignments and Textual Conventions

Manageable Object Definitions

Trap Definitions

Where SNMP Components Reside

SNMP Management components reside on:

A remote client (or Network Management Station)

A network device

Remote client components include the network manager and the MIBs. The network manager gets query information from the MIB to send a request to the network device. The interaction between the manager and MIB is internal to the network manager and specific to each network manager. A number of network managers are available commercially and each have their own method for interaction among MIBs or other management strategies. Some network managers on the network support management tools other than SNMP.

Network device components include the SNMP agent and the Instrumentation Agents (IA). Routers, servers, gateways, and printers are all examples of different network device types that support SNMP.

The SNMP agent receives a request and passes it on to the appropriate IA. The IA requests the information from the internals of the network device. The response is sent to the SNMP agent, which sends it through the network to the network manager.

Parent Topic:

Functional SNMP Components

Related Topics:

SNMP Operation

Management Information Bases

Instrumentation Agents

NWSNMP: Functions

OCBAbortSet

Defines a function to abort a previously checked Set request
NetWare Server:
Platform: NLM
Service: NWSNMP

Syntax

```
void OCBAbortSet (
   struct ObjControl  *ocb_pt,
   AttrValue          *inst_list,
   void               *handle_pt);
```

Parameters

ocb_pt

Points to the ObjControl structure containing the Object Control Block that called **OCBAbortSet**.

inst_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks that give the index values of the object instance (NULL if the specified object has no instance information).

handle_pt

Points to a handle returned by the **OCBCheckSet** function.

Remarks

The function defined by **OCBAbortSet** is called when a Set request was checked by calling the **OCBCheckSet** function but the defined function cannot be completed. For example, the Set request might be aborted because the NWSNMP request makes several Set requests on different objects and one of the Set requests cannot be completed for another object.

If cleanup is not necessary when a Set request aborts, you do not need to call **OCBAbortSet**. Set the *ocb_abort_set* field of the ObjControl structure to NULL.

Your exporting module must call the **OCBAbortSet** function when cleanup is necessary. Set the *ocb_abort_set* field of the ObjControl structure to the address of the **OCBAbortSet** function for each abstract object.

Your function must free any resources that were allocated by the **OCBCheckSet** function in preapration for handling the Set request. The *handle_pt* parameter, originally set in the **OCBCheckSet** function, provides a pointer to these resources.

See Also

OCBCheckSet

OCBCheckSet

Defines a function to ensure a Set request is legal

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int OCBCheckSet (
   struct ObjControl  *ocb_pt,
   AttrValue          *inst_list,
   AttrValue          *attr_list,
   void              **handle_pp);
```

Parameters

ocb_pt

Points to the ObjControl structure containing the Object Control Block that called **OCBCheckSet**.

inst_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks specifying the index values of the object instance (NULL if the object has no instance information).

attr_list

Points to the AttrValue structure containing a NULL-terminated list of the Attribute Value Blocks for the attributes being set by the request.

handle_pp

Points to a pointer to a buffer into which **OCBCheckSet** can return a handle to be passed to either the **OCBCommitSet** or **OCBAbortSet** function.

Return Values

SNMP_ERR_NOERROR	Successful
SNMP_ERR_*	Describes why values cannot be set. When the SNMP_ERR_* failure is caused by a particular attribute, set the *aib_info* field to NULL and set the *aib_length* field to the returned error code for the attribute.

Remarks

OCBCheckSet determines whether your exporting module allows the setting of a set of attributes in a specific object instance. During a Set request, **OCBCheckSet** determines whether all the requested attributes of all the requested objects can be legally set before actually changing the data store. Once NWSNMP has received a successful reutrn from all object instances in the request, **OCBCommitSet** is called for each object instance to make the changes.

You do not need to call **OCBCheckSet** if Set requests always succeed on the specified object. Otherwise, you must call the **OCBCheckSet** function. Set the *ocb_check_set* field of the ObjControl structure to the address of the **OCBCheckSet** function for each abstract object.

Your function should complete the following steps:

1. Locate the object instance pointed to by the index information in the Attribute Value Blocks pointed to by the *inst_list* parameter.

2. Confirm that the set of attributes of the specific object instance can legally be set to the given values without modifying the database. Your application can block, but it should only block if necessary.

   IMPORTANT: Do not block in the **OCBCheckSet** function unless absolutely necessary. Because a Set request affects multiple objects and changes the system state, the transition window for completing the Set request must be as small as possible. If your function blocks, there could be a change in the system state on which another affect object depends. The system state could then become inconsistent once the Set request is complete.

3. Allocate any resources you will subsequently need to complete the Set request when NWSNMP calls the **OCBCommitSet** function. This ensures that the **OCBCommitSet** function does not fail from lack of resources.

   If you allocate resources, associate all the resources with a context block, and place a pointer to the context block in the *handle_pp* parameter. NWSNMP returns the *handle_pp* parameter to your NLM when it calls the **OCBCommitSet** or **OCBAbortSet** function.

   If no resources are needed, set the *handle_pp* parameter to NULL.

   IMPORTANT: If you have already allocated some resources and the check subsequently fails, clean up the previously allocated resources before returning. NWSNMP does not call the **OCBAbortSet** function if your **OCBCheckSet** function fails.

4. Set the appropriate return value.

See Also

OCBCommitSet

OCBCommitSet

Defines a function that sets a list of attributes in an object

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int OCBCommitSet (
   struct ObjControl  *ocb_pt,
   AttrValue          *inst_list,
   AttrValue          *attr_list,
   void               *handle_pt);
```

Parameters

ocb_pt

Points to the ObjControl structure containing the Object Control Block that called **OCBCommitSet**.

inst_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks that give the index values of the object instance (NULL if the object has no instance information).

attr_list

Points to the AttrValue structure containing a NULL-terminated list of hte Attribute Value Blocks for the attributes being set by the request.

handle_pt

Points to a handle returned by the **OCBCheckSet** function.

Return Values

SNMP_ERR_NOE RROR	Successful
SNMP_ERR_*	Describes why values cannot be set. When the SNMP_ERR_* failure is caused by a particular attribute, set the *aib_info* field to NULL and set the *aib_length* field to the returned error code for the attribute.

Remarks

NWSNMP calls your function defined by **OCBCommitSet** to accomplish an SNMP Set request. NWSNMP calls your function only after it has called the **OCBCheckSet** function for all object instances to ensure that the Set request can be legally completed.

Your exporting module must call the **OCBCommitSet** function for each object that has one or more writeable attributes. Set the *ocb_commit_set* field of the ObjControl structure to the address of the **OCBCheckSet** function for each abstract object.

Your function must complete the following steps:

1. Use the attribute list pointed to by the *inst_list* parameter to find the object instance being addressed.

2. Update the specific object instance to reflect the values in the list of attribute values pointed to by the *attr_list* parameter.

   The handle_pt parameter, originally set in the OCBCheckSet function, provides a pointer to any resources required for the Set request. This ensures that OCBCommitSet does not fail because it cannot access required resources.

   IMPORTANT: Do not block in **OCBCommitSet** unless absolutely necessary. Because a Set request affects multiple objects and changes the system state, the transition window for completing the Set request must be as small as possible. If your function blocks, there could be a change in the system state on which another affect object depends. The system state could then become inconsistent once the Set request is complete.

3. Set the appropriate return value.

   IMPORTANT: **OCBCommitSet** should never fail. Any circumstances that might cause a Set request to fail must be handled in the **OCBCheckSet** function.

See Also

OCBCheckSet

OCBGet

Defines a function used to read attributes from a specific object

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int OCBGet (
   struct ObjControl  *ocb_pt,
   AttrValue          *inst_list,
   AttrValue          *attr_list);
```

Parameters

ocb_pt

Points to the ObjControl structure containing the Object Control Block that called **OCBGet**.

inst_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks that give the index values of the object instance.

attr_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks specifying the attributes to be read.

Return Values

SNMP_ERR_NOERROR	Successful
SNMP_ERR_*	Describes why values cannot be set. When the SNMP_ERR_* failure is caused by a particular attribute, set the *aib_info* field to NULL and set the *aib_length* field to the returned error code for the attribute.

Remarks

OCBGet is a typedef function definition. Your exporting module must call the **OCBGet** function for each abstract object. Set the *ocb_get* field of the ObjControl structure to the address of the **OCBGet** function for each abstract object.

If the function fails for any SNMP object in the request, the entire request fails.

Your function must complete the following steps:

1. Locate the correct instance of the abstract object by using the index information in the NULL-terminated list of Attribute Value Blocks pointed to by the *inst_list* parameter.

2. Search through the NULL-terminated attribute list of the Attribute Value Blocks pointed to by the *attr_list* parameter and copy the value of each attribute into its block.

   If the AIF_IMMEDIATE flag is set for the attribute, copy the value into the *avb_length* field. Otherwise, copy the value into the buffer pointed to by the *avb_value* field and set the avb_length field to the length (in bytes) of the value. If the buffer is too small, return SNMP_ERR_TOOBIG for the attribute.

3. Return an appropriate return value.

See Also

OCBGetNext

OCBGetNext

Defines a function used to read attributes from the object instance that follows the specified object instance

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int OCBGetNext (
   struct ObjControl  *ocb_pt,
   AttrValue          *inst_list,
   AttrValue          *next_list,
   AttrValue          *attr_list);
```

Parameters

ocb_pt

Points to the ObjControl structure containing the Object Control Block that called **OCBGetNext**.

inst_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks that give the index value of the current object instance.

next_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks in which to place the index value of the next object instance.

attr_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks containing the attributes to be read once the next instance is located.

Return Values

SNMP_ERR_NOERROR	Successful
SNMP_ERR_*	Describes why values cannot be set. When the SNMP_ERR_* failure is caused by a particular attribute, set the *aib_info* field to NULL and set the *aib_length* field to the returned error code for the attribute.

SNMP_ERR_NOSUCH NAME	No next instance of the object. Do not flag any attributes.

Remarks

The function defined by **OCBGetNext** should fill in the Attribute Value Blocks pointed to by the *attr_list* parameter with the values of the attributes to be read.

Your exporting module should call the OCBGetNext function for any object with multiple instances. Set the *ocb_get_next* field of the ObjControl structure to the address of **OCBGetNext** for each abstract object.

NOTE: You do not need to call **OCBGetNext** for objects with only one instance. NWSNMP calls the **OCBGet** function to fulfill GetNext requests for these objects. For single-instance objects, set the *ocb_get_next* field to NULL.

If the function fails for any SNMP object in the request, the entire request fails.

Your function should complete the following steps:

1. Locate the previous instance of the abstract object by using the index information in the Attribute Value Blocks pointed to by the *inst_list* parameter.

2. Locate the next object instance. The exact meaning of “next” depends on the MIB definition of the object. In general, the objects are presented in lexicographical order of their MIB INDEX values.

   If there is no next instance, return SNMP_ERR_NOSUCHNAME. NWSNMP will then locate the next value after your object.

3. Return the index information for the new instance ID in the Attribute Value Blocks pointed to by the *next_list* parameter. Specify the indexing attributes in the same order as in the INDEX clause of the object MIB.

4. Search through the list of Attribute Value Blocks pointed to by the *attr_list* parameter and use the next instance of the object to write the correct value of the attribute into each Attribute Value Block.

   If the AIF_IMMEDIATE flag is set for the attribute, copy the value into the *avb_length* field. Otherwise, copy the value into the buffer pointed to by the *avb_value* field and set the *avb_length* field to the actual length (in bytes) of the value. If the buffer is too small, return SNMP_ERR_TOOBIG for the attribute.

5. Return an appropriate return code.

See Also

OCBGet

OCBLookupInst

Defines a function used to find an instance from an instance group that has been instrumented using the standard access methods

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
unsigned long *OCBLookupInst (
   struct ObjControl  *ocb_pt,
   AttrValue          *inst_list,
   AttrValue          *next_list);
```

Parameters

ocb_pt

Points to the ObjControl structure containing the Object Control Block that called **OCBLookupInst**.

inst_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks that give the index values of the current instance array within the group table (NULL if the object has no instance information).

next_list

Points to the AttrValue structure containing a NULL-terminated list of Attribute Value Blocks in which to place the index values of the next instance array within the group table.

Return Values

Returns a pointer to an array of unsigned long values.

Remarks

OCBLookupInst should fill in the Attribute Value Blocks pointed to by the *next_list* parameter in the same order as in the INDEX clause of the object MIB.

If the *next_list* parameter is NULL, the function is treated as a Get request instead of a GetNext request.

OCBLookupInst returns a pointer to a row that is a simple table.

Your exporting module must call **OCBLookupInst** for any group table you define. The SAI_DEFINE_GROUP macro places a pointer to **OCBLookupInst** in the *ocb_context_pt* field of the Object Control Block.

Your function should complete the following steps:

1. Locate the simple table instance pointed to by the *inst_list* parameter.

2. If the *next_list* parameter is NULL, a Get request should be called. The index described the *inst_list* parameter must exactly match the index of one of the simple tables in the group table. Return the pointer to the simple table described by the Attribute Value Blocks as pointed to by the *inst_list* parameter. If there is no match, return SNMP_ERR_NOSUCHNAME.

3. If the *next_list* parameter is not NULL, a GetNext request should be called. Locate the next simple table following the simple table described by the *inst_list* parameter. Return the pointer to this table. Place the index information for the table in the Attribute Value Blocks pointed to by the *next_list* parameter in the same order as in the INDEX clause of the object MIB.

   If you cannot find the desired instance, return NULL.

See Also

SAI_DEFINE_TABLE

SACReadAttributes

Reads attributes from an abstract object

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int SACReadAttributes (
   unsigned long  *object_id_pt,
   int             inst_ct,
   struct TLV     *inst_pt,
   int             attr_ct,
   struct TLV     *attr_pt);
```

Parameters

object_id_pt

Points to the beginning of a NULL-terminated array of long word types containing the abstract object ID.

inst_ct

Specifies the number of type/length/values in the instance ID array pointed to by the *inst_pt* parameter.

inst_pt

Points to the TLV structure containing an array of type/length/values identifying the object instance (NULL if the *inst_ct* parameter is zero).

attr_ct

Specifies the number of type/length/values in the attribute buffer array pointed to by the *attr_pt* parameter.

attr_pt

Points to the TLV structure containing an array of type/length/value structures containing the attributes to read and the buffer space in which the values should be returned (NULL if the *attr_ct* parameter is zero).

Return Values

SNMP_ERR_NOERROR	Successful
error code	Unsuccessful

Remarks

SACReadAttributes can block before returning.

A management application can call SACReadAttributes to extract information from the management entities that have registered objects with NWSNMP.

If the object being requested has a single instance only, the *inst_ct* parameter can be zero and the *inst_pt* parameter can be NULL.

If no attributes are desired, the *attr_ct* parameter can be zero and the *attr_pt* can be NULL.

SACReadAttributes finds the abstract object in the SNMP database. If then calls the **OCBGet** function for the object to read the indicated attributes from the object instance.

The *inst_pt* parameter array provides the necessary index information to identify the specific object instance, as specified in the SNMP MIB definition for the object. The instance values must be in the same order as they are found in the INDEX clause of the object MIB. NWSNMP ignores the *tlv_id* fields in the elements of the array.

In each type/length/value structure in the *attr_pt* parameter array, you should complete the following steps:

1. Set the *tlv_id* field to the ID of the attribute to be read.
2. Set the *tlv_value* field to point to a buffer where the value should be returned.
3. Set the tlv_length field to the number of bytes in the buffer. (NWSNMP writes the actual length of the value into the *tlv_length* field.)

If NWSNMP finds a non-zero *tlv_type* field when processing the *attr_pt* parameter array, it checks the actual type of the attribute to determine whether it agrees with the *tlv_type* field. If the two types disagree, **SACReadAttributes** fails. If the *tlv_type* field is zero, NWSNMP writes the actual SNMP type code for the attribute into the field.

See SACReadNextAttributes: Example.

SACReadSNMPObject

Returns the SNMP object value for an SNMP object ID

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int SACReadSNMPObject (
   int              object_id_ln,
   unsigned long   *object_id_pt,
   struct TLV      *value_buffer);
```

Parameters

object_id_ln

Specifies the number of long word types in the SNMP object ID pointed to by the *object_id_pt* parameter.

object_id_pt

Points to the beginning of a NULL-terminated array of unsigned long integers containing the SNMP object ID.

value_buffer

Points to the TLV structure containing a description of the buffer to which NWSNMP should write the SNMP object value.

Return Values

SNMP_ERR_NOERROR	Successful
error code	Unsuccessful

Remarks

You must know the specific SNMP object ID to call **SACReadSNMPObject**.

SACReadSNMPObject can block before returning.

SACReadSNMPObject is not subject to authentication. You are responsible for ensuring the information does not get read by unauthorized requests.

The *tlv_value* field should point to a buffer to receive the returned value, and the *tlv_length* field should contain the length of the buffer.

SACReadSNMPObject parses the object ID and calls the appropriate management entity to retrieve the object value. NWSNMP then completes the following steps:

1. Writes the resulting value into the memory pointed to by the *tlv_value* field.

2. Sets the *tlv_length* field to the actual length of the value.

3. Sets the *tlv_id* field to the attribute ID of the attribute read (not expected to be useful in this context).

4. Checks the *tlv_type* field to determine whether it is zero and continues processing the request.

   If the *tlv_type* field is zero, NWSNMP sets it to the correct type.

   If the *tlv_type* field is non-zero, NWSNMP checks to ensure that the type agrees with the actual type of the value. **SACReadSNMPObject** fails is the two types do not agree.

See SACReadSNMPObject: Example.

SACSetAttributes

Sets specified attribute values for an abstract object

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
int SACSetAttributes (
   unsigned long  *object_id_pt,
   int             inst_ct,
   struct TLV     *inst_pt,
   int             attr_ct,
   struct TLV     *attr_pt);
```

Parameters

object_id_pt

Points to the beginning of a NULL-terminated array of long word types containing the abstract object ID.

inst_ct

Specifies the number of type/length/values in the instance ID array pointed to by the *inst_pt* parameter.

inst_pt

Points to the TLV structure containing an array of type/length/values which identify the object instance (NULL if the *inst_ct* parameter is zero).

attr_ct

Specifies the number of type/length/values contained in the attribute value array pointed to by the *attr_pt* parameter.

attr_pt

Points to the TLV structure containing an array of type/length/value structures describing the attribute values to set (NULL is the*attr_ct* parameter is zero).

Return Values

SNMP_ERR_NOERROR	Successful
error code	Unsuccessful

Remarks

An application can call **SACSetAttributes** to set information in the objects that various management entities have registered with NWSNMP.

SACSetAttributes can block before returning.

SACSetAttributes finds the abstract object in its database. It then calls the **OCBCheckSet** and **OCBCommitSet** functions for the object to set the indicated attributes in the object instance.

If the object being modified has a single instance only, the *inst_ct* parameter can be zero and the *inst_pt* parameter can be NULL.

If no attributes should actually be modified, the *attr_ct* parameter can be zero and the *attr_pt* parameter can be NULL.

The *inst_pt* parameter array provides the necessary index information to identify the specific object instance as specified in the object SNMP MIB definition. The instance values must be in the same order as they are found in the INDEX clause of the object MIB. NWSNMP ignores the *tlv_id* fields in the elements of the array.

The *attr_pt* parameter array indicates which attributes should be set and to which value they should be set. In each type/length/value structure, you should set the *tlv_id* field to the ID of the attribute to set. Set the *tlv_value* field to the value to set and the *tlv_length* field to the number of bytes in the value.

If NWSNMP finds a non-zero *tlv_type* field when processing the *attr_pt* parameter array, it checks the actual type of the attribute to determine whether the *tlv_type* field agrees. If the two types disagree, **SACSetAttributes** fails.

If the *tlv_type* field is zero, NWSNMP writes the actual SNMP type code for the attribute set into the field.

See SACSetAttributes mib-2: Example and SACSetAttributes on Table Object: Example.

SACTrap

Generates an SNMP trap

NetWare Server:

Platform: NLM

Service: NWSNMP

Syntax

```
void SACTrap (
   unsigned long   *enterpriseId,
   int              genericType,
   int              specificType,
   int              varBindCount,
   struct VarBind  *varBinds,
   char            *communityString);
```

Parameters

enterpriseId

Specifies the enterprise ID of the MIB generating the enterprise-specific trap (must be NULL-terminated).

genericType

Specifies the generic type of the trap (always SNMP_TRAP_ENTERPISESPECIFIC).

specificType

Specifies the specific type of the trap (defined in the associated trap MIB).

varBindCount

Specifies the number of variable bindings specified for the trap.

varBinds

Points to the VarBind structure containing an array identifying the variable instance and value information to be included in the trap.

communityString

Points to a NULL-terminated octet string to be used as the community name for the trap (NULL if the default trap community name is used).

Remarks

Any entity in the system can generate an SNMP trap by calling **SACTrap** .

SACTrap takes the trap type information and the specified variable bindings and combines them with information internal to NWSNMP to bindings and combines them with information internal to NWSNMP to produce an SNMP trap message as described by the SNMP standard. It then sends the trap message to each SNMP manager configured to receive traps for the indicated community.

Do not call **SACTrap** to generate SNMP generic traps.

SACTrap can block before returning.

The *specificType* parameter should contain an application-specific trap type.

The variable bindings pointed to by the *varBinds* parameter are specific to the type of trap being generated.

See SACTrap: Example.

SAIDeregisterMIB

Deregisters a B definition with NWSNMP
NetWare Server:
Platform: NLM
Service: NWSNMP

Syntax

```
void SAIDeregisterMIB (
   ObjInfo **mib_pp);
```

Parameters

mib_pp

Points to the ObjInfo structure containing an array of pointers to Ojbect Information Blocks specifying the MIB handle exported from the compiled MIB definition

Remarks

SAIDeregisterMIB allows an exiting NLM to remove its MIB information from the NWSNMP data store.

Any NLM that called the **SAIRegisterMIB** function to register a MIB with NWSNMP must call **SAIDeregisterMIB**. The NLM typically deregisters the MIB when it is exiting because it will no longer be available to support the MIB.

The MIB handle identifies the MIB tree compiled by the MIB compiler during the build process.

When an NLM calls **SAIDeregisterMIB**, NWSNMP forgets about the part of the MIB tree described by the MIB handle. **SAIDeregisterMIB** must be called before the NLM supporting the object unloads.

See SAIDeregisterMIB: Example.

See Also

SAIRegisterMIB

SAIRegisterMIB

Allows a management entity to register its MIB tree with NWSNMP
NetWare Server:
Platform: NLM
Service: NWSNMP

Syntax

```
int SAIRegisterMib (
   ObjInfo       **mib_pp,
   unsigned long   rtag);
```

Parameters

mib_pp

Points to the MIB handle exported from the compiled MIB definition (located in the header file produced by the MIB compiler).

rtag

Specifies the resource tag to which the MIB should be charged.

Return Values

SAI_REGISTER_OK	Successful
SAI_REGISTER_FAIL_DUP	One of the abstract objects is already registered.
SAI_REGISTER_BAD_RTAG	The resource tag is not for objects of type SAI_MIB_RTAG.

Remarks

NWSNMP MIB resource tags use the ID SAI_MIB_RTAG defined by the SNMP interface header file agent.h to define the *rtag* parameter.

The NLM that wants to present a MIB through SNMP calls **SAIRegisterMIB** to inform NWSNMP of the part of the MIB tree it is processing.

At the leaves of the MIB tree are the MIB Object Information Blocks that refer to the Object Control Blocks defined by the NLM support modules.

Call the NetWare C Library **AllocateResourceTag** function to allocate a resource tag for your object.

See SAIRegisterMIB: Example.

See Also

SAIDeregisterMIB

NWSNMP: Structures

AttrInfo

Points to the Attribute Information Block that describes the characteristics of a specified attribute

Service: NWSNMP

Defined In: nwmediam.h

Structure

```
typedef struct {
   int   aib_id;
   int   aib_type;
   int   aib_asn_type;
   int   aib_min;
   int   aib_max;
   int   aib_flags;
} AttrInfo;
```

Fields

aib_id

Specifies the attribute ID for the specific attribute.

aib_type

Specifies the data type of the specific attribute.

aib_asn_type

Specifies the type of the specific attribute (see sa_asn1.h for values).

aib_min

Specifies the minimum legal value for an integer type or the smallest legal length for an octet string or display string type.

aib_max

Specifies the maximum legal value for an integer type or the longest legal length for an octet string or display string type.

aib_flags

Specifies the flags to use.

Remarks

The *aib_id* field identifies the object variable in the MIB tree.

The *aib_type* field can have the following values:

Name	Va lue	Description

AIT_UNKNOWN	0	Do not use in the AttrInfo structure.
AIT_INTEGER	1	Integer. Can specify a guage, counter, etc..
AIT_TIMETICKS	2	Each timetick represents 1/100 of a second.
AIT_SHORTOCTET	3	Octet strings of 4 bytes or less.
AIT_LONGOCTET	4	Octet strings of more than 4 bytes.
AIT_DISPLAYSTRING	5	NVT ASCII strings.
AIT_OBJECTID	6	An object ID.
AIT_CHOICE	7	ASN.1 is standard. Or the choice value (a union).
AIT_HEADER	8	Type and length without any value (not used in actual attribute blocks).

When the attribute is encoded as an ASN.1 value, the *aib_asn_type* field specifies the ASN.1 type. The*aib_asn_type* field is usually of interest only to NWSNMP.

The *aib_min* and *aib_max* fields are ignored unless the AIF_LIMITS flag is set.

The *aib_flags* field can have the following values:

Name	**Description**
AIF_READ	Attribute can be read by a sufficiently authorized request. Does not indicate whether the current request is authorized.
AIF_WRITE	Attribute can be written by a sufficiently authorized request. Does not indicate whether the current request is authorized.
AIF_IMMEDIATE	Attribute fits in an unsigned long. These attributes are always located in the *avb_length* field of the Attribute Value Block instead of the associated *avb_value* field buffer which eliminates the need for a buffer for values that are four bytes or less in length.
AIF_LIMITS	Attribute has a limited range or size.

Each Attribute Value Block points to the Attribute Information Block of the corresponding attribute to provide identification and treatment information for the specific attribute value carried in the AttrValue structure block.

The MIB compiler creates one of these blocks for each attribute in the MIB.

> **IMPORTANT:** Do not change the blocks or modify the information within them.

The Attribute Value Block is used by NWSNMP to process requests for each attribute. NWSNMP converts the values and checks the range and access so those fields of the Attribute Information Block are not useful to the access routines.

For the convenience of the object-handling functions, NWSNMP passes a pointer to the Attribute Information Block in the*aib_info* field of the AttrValue structure.

AttrValue

Passes information between NWSNMP and an NLM during Get, GetNext, and Set functions

Service: NWSNMP

Structure

```
typedef struct {
   struct AttrValue  *avb_next;
   unsigned long      avb_length;
   void              *avb_value;
} AttrValue;
```

Fields

avb_next

Points to the next attribute value in the list (pass NULL if the attribute is last in the list).

avb_length

Specifies the length, in bytes, of the buffer pointed to by the*avb_value* field.

avb_value

Points to a buffer containing the attribute value.

Remarks

The Attribute Value Blocks are always in the same order and have the same number of values as in the INDEX clause of the object MIB.

If the AIF_IMMEDIATE flag is set in the Attribute Information Block for the specified attribute, the*avb_length* field specifies the attribute value and the *avb_value* field will be ignored.

NWSNMP passes a NULL-terminated list of Attribute Information Blocks to each object-handling function to provide the function with the attribute being accessed. NWSNMP uses the following types of Attribute Information Block arrays:

Name	Description
instance list	The object-handling function uses this set of Attribute Information Blocks to identify the index values of the object instance to be accessed. The values are always given in the same order as in the INDEX clause of the object MIB.
next list	The **OCBGetNext** and **OCBLookupInst** functions use this

	set of Attribute Information Blocks to identify the next object instance once it is located. The values are always given in the same order as in the INDEX clause of the object MIB.
attribute list	The object-handling functions use this set of Attribute Information Blocks to identify specific attributes and their values. The functions either extract each attribute value from each block (Set functions) or write the value into each block (Get and GetNext functions).

NWSNMP passes lists of Attribute Value Blocks to the object-handling functions. The number of lists that are passed depends on the specific function. The object-handling functions are responsible for copying attribute values into and out of the Attribute Value Blocks that NWSNMP allocates.

IMPORTANT: All attributes that have the AIF_IMMEDIATE flag set are copied directly from or into the *avb_length* field. Attributes that are not immediate must be copied from or into the buffer pointed to by the *avb_value* field. When an object-handling function is copying data from the buffer, the *avb_length* field gives the actual length (in bytes) of the entire value.

When a object-handling function is expected to return data in the *avb_value* field, NWSNMP initially sets the *avb_length* field to the length (in bytes) of the entire buffer. Once the function copies the data into the buffer, it should update the *avb_length* field to reflect the actual length of the attribute value. If the buffer is too short for the attribute value, the function should return SNMP_ERR_TOOBIG.

ObjControl

controls the requests that can be applied to an abstract object

Service: NWSNMP

Structure

```
typedef struct {
   unsigned int   ocb_flags;
   void          *ocb_context_ptr;
   OCBGet         ocb_get;
   OCBGetNext    *ocb_get_next;
   OCBCheckSet   *ocb_check_set;
   OCBCommitSet  *ocb_commit_set;
   OCBAbortSet   *ocb_abort_set;
} ObjControl;
```

Fields

ocb_flags

Specifies the flags that modify normal object treatment (no flags are currently defined).

ocb_context_pt

Points to the context and used by the standard access functions.

ocb_get

Specifies the function that will return attribute values out of an object by passing an object ID/instance ID pair.

ocb_get_next

Specifies the function that will return attribute values out of the next object by passing the object ID/instance ID pair of the previous object.

ocb_check_set

Specifies the function that will check whether a Set request on a specific set of attributes can be legally completed for the specified object (NULL if Set requests always succeed).

ocb_commit_set

Specifies the function that will complete a previously checked Set request for a specific object (NULL if the object is not writeable).

ocb_abort_set

Specifies the function that will abort a Set request that has successfully called the function specified by the *ocb_check_set* field, but which cannot be completed for other reasons (NULL if the object requires no cleanup when the Set request is aborted).

Remarks

The exporting module defines the Object Control Block for each abstract object and NWSNMP uses the block to perform requests on the abstract object. NWSNMP passes the Object Control Block to functions and provides them with the necessary context to complete each request.

The *ocb_context_pt* field can have the following values:

SAI_DEFINE_TABLE	Points to the table location.
SAI_DEFINE_POINTER	Points to the table location.
SAI_DEFINE_GROUP	Points to the **OCBLookupInst** function for the group table.

Use the Object Control Block in the NLM that exports the abstract object and describes the functions that NWSNMP should call to manipulate attributes of a specified object. The Object Control Block must be initialized at link or run time (before the MIB handle is passed to the **SAIRegisterMIB** function).

IMPORTANT: Once the NLM registers the MIB with NWSNMP, do not modify the Object Control Block.

For Get requests, NWSNMP calls the function pointed to by the *ocb_get* field. For GetNext requests, NWSNMP calls the function pointed to by the *ocb_get_next* field.

For Set requests, NWSNMP calls the function pointed to by the *ocb_check_set* field. If that function returns successfully for all objects, NWSNMP will call the function pointed to by the *ocb_commit_set* field. If the function pointed to by the *ocb_check_set* field fails for any object, NWSNMP will call the function pointed to by the *ocb_abort_set* field for all objects for which the check routine was called successfully.

Individual NWSNMP requests can operate on more than one abstract object and can apply to several attributes for each object. NWSNMP calls the object-handling functions exactly once for each object in the request and combines all attributes for that object into a single request.

For example:

```
ObjControl OCB_ip {
   OCBF_NONE,   /*no flags*/
   ipMIB,       /*pointer to the statistics array context*/
   IPStatGet,   /*based on OCBGet function*/
   NULL,        /*no GetNext is necessary*/
   NULL,        /*Set requests always succeed; do not check*/
   IpStatSet,   /*based on OCBCommitSet function*/
```

```
      NULL          /*no check so no abort either*/
   };
```

ObjInfo

Describes the object to NWSNMP
Service: NWSNMP

Structure

```
typedef struct {
   struct ObjInfo                *oib_next;
   ObjControl                    *oib_control;
   struct ResourceTagStructure  *oib_rtag;
   unsigned int                  oib_id_ln;
   unsigned long                *oib_id_pt;
   unsigned int                  oib_index_ln;
   AttrInfo                     *oib_index_pt;
   unsigned int                  oib_attr_mx;
   AttrInfo                     *oib_attr_pt;
} ObjInfo;
```

Fields

oib_next

Points to the next object in the NWSNMP list of registered objects.

oib_control

Points to the Object Control Block.

oib_rtag

Points to the resource tag used to register the specified object.

oib_id_ln

Specifies the number of unsigned long values in the object ID array.

oib_id_pt

Points to the object ID.

oib_index_ln

Specifies the number of items in the index for the specified object (0 specifies an unindexed object).

oib_index_pt

Points the AttrInfo structure containing a list of attribute descriptions in the instance index (0 specifies the object has only one instance).

oib_attr_mx

Specifies the number of attributes for the specified object.

oib_attr_pt

Points to the AttrInfo structure containing an array of attribute descriptions for each attribute of an object (indexed by attribute number).

Remarks

An Object Information Block is generated by the MIB compiler for each object and is used internally by NWSNMP.

WARNING: The Object Information Block is used internally by NWSNMP. Do not modify the block.

TLV

describes an arbitrary value by providing ASN.1 identification, type, length, and location for the value

Service: NWSNMP

Structure

```
typedef struct {
   int    tlv_id;
   int    tlv_type;
   int    tlv_length;
   void  *tlv_value;
} TLV;
```

Fields

tlv_id

Specifies the identifier describing what the value actually means (optional).

tlv_type

Specifies the appropriate ASN.1 type of the value (if known).

tlv_length

Specifies the total number of bytes in the value (or the total number of bytes available to receive the value).

tlv_value

Points to the value or to a buffer to receive the value.

Remarks

The Get and Set functions in the client interface use the TLV structure to describe values. NWSNMP copies the value information into and out of the TLV structure buffer.

The NLM that accesses the NWSNMP data must define and provide the appropriate TVL structures.

VarBind

Service: NWSNMP

Structure

```
typedef struct {
   unsigned long  *vb_id_pt;
   int             vb_id_ln;
   int             vb_encoding;
   int             vb_asn_type;
   unsigned long   vb_length;
   void           *vb_value;
} VarBind;
```

Fields

vb_id_pt

Points to an array of unsigned long data containing the variable binding object ID.

vb_id_ln

Specifies the number of sub IDs in the object ID.

vb_encoding

Specifies the AIT_* value indicating the appropriate encoding mechanism for handling the specified value.

vb_asn_type

Specifies the value to put in the type field of the encoded ASN.1 type/length/value.

vb_length

Specifies the number of bytes in the value or the immediate value of the variable bind for AIT_* values.

vb_value

Points to the value (if not an immediate value).

Remarks

The trap generation function in the client interface uses the VarBind structure to describe variable bindings. NWSNMP copies the binding information into and out of buffers described by the VarBind structure.

The NLM that accesses the NWSNMP data must define and provide the appropriate VarBind structures.

NWSNMP: Macros

SAI_DEFINE_GROUP

Is a macro that defines a group table-oriented object
NetWare Server:
Platform:
Service: NWSNMP

Syntax

```
OCBLookupInst  Lookup_Routine;

SAI_DEFINE_GROUP (
   Object_Name,
   Lookup_Routine);
```

Parameters

Ojbect_Name

Specifies the name of the Object Control Block being declared (should be the MIB ‘OCB_’ object name as required by the MIB compiler).

LookupRoutine

Specifies the **OCBLookupInst** function that returns an array arranged in the format of a simple table from which actual values can be read or written.

Remarks

SAI_DEFINE_GROUP defines an Object Control Block with the name specified by the *Object_Name* parameter that provides access to the group table. The table is defined as a group of instances, where each individual instance is arranged as a simple array of SNMP values.

Your exporting NLM should call **SAI_DEFINE_GROUP** to instrument a group table. You must call the **OCBLookupInst** function that accesses the group table and place a pointer to the **OCBLookupInst** function in the *Lookup_Routine* parameter.

The **OCBLookupInst** function must be able to find a given table instance and return a pointer to that table.

The elements of a group table are a standard format table in the simple table format for which standard access functions can be called.

See Also

SAI_DEFINE_TABLE

SAI_DEFINE_POINTER

Is a macro that defines a pointer table-oriented object
NetWare Server:
Platform:
Service: NWSNMP

Syntax

```
unsigned long *Table_Location;

SAI_DEFINE_POINTER (
   Object_Name,
   Table_Location);
```

Parameters

Object_Name

Specifies the name of the Object Control Block being declared (the MIB "OCB_" object name as required by the MIB compiler).

Table_Location

Specifies the name of an array of pointers defining the table.

Remarks

SAI_DEFINE_POINTER defines an Object Control Block with the name specified by the *Object_Name* parameter and provides access to the pointer table starting at the location specified by the *Table_Location* parameter.

Your exporting NLM should call **SAI_DEFINE_POINTER** to instrument a previously created pointer table that is formatted as an array of pointers.

NOTE: The first element of the array (at index zero) should be the number of elements in the array. The type of the first element is an unsigned long and not a pointer. NWSNMP uses this value to determine whether the table supports all the attributes in the MIB.

The array specified by the *Table_Location* parameter can contain both long values and various kinds of pointers since unsigned long values and pointers are assumed to be the same size.

Depending on the MIB definition, each element of the array can contain any one of the following type of attributes in the prescribed form:

Integer	Contained in the array as a pointer to an unsigned long.
TimeTicks	Contained in the array as a pointer to an unsigned long. The unit of time in this value must be NetWare operating system ticks.
short OCTET STRING	Contained in the array as a pointer to an unsigned long with the low order byte being the first octet of the string.
long OCTET STRING	Contains in the array as a pointer to an array of type char that contains the value. The value must be a NULL-terminated string. Any long OCTET STRING that can contain NULL must be specially handled by the support module of the object. If the string is writeable, the pointer must point to a buffer large enough to handle the maximum length value. Any long OCTET STRING that has no maximum length in the MIB is assumed to have a maximum of 256 characters, including the terminating NULL. If a truly unbound, writeable OCTET STRING must be supported by the object, the object support code must specially handle it.
OBJECT ID	Contained in the array as a pointer to an array of type unsigned long. The OBJECT ID is terminated by an element of zero. If the OBJECT ID is writeable, the pointer must point to a buffer large enough to handle a 256-element OBJECT ID. If a truly unbound, writeable OBJECT ID must be supported by the object, the object support code must specially handle it.

The format of this table and, consequently, the implementation of **SAI_DEFINE_POINTER** and its supporting functions can vary in different architectures that have different size pointers and unsigned long values.

The unit of time for TimeTicks can vary between environments.

See Also

SAI_DEFINE_TABLE

SAI_DEFINE_TABLE

Is a macro that defines a simple table-oriented object
NetWare Server:
Platform:
Service: NWSNMP

Syntax

```
unsigned long Table_Location;

SAI_DEFINE_TABLE (
   Object_Name,
   Table_Location);
```

Parameters

Object_Name

Specifies the name of the Object Control Block being declared (the MIB "OCB_" object name as required by the MIB compiler).

Table_Location

Specifies the name of an array of unsigned long values defining the table.

Remarks

SAI_DEFINE_TABLE defines an Object Control Block with the name specified by the *Object_Name* parameter and provides access to the simple table starting at the location specified by the *Table_Location* parameter.

Your exporting NLM should call **SAI_DEFINE_TABLE** to instrument a previously created simple table formatted as an array of unsigned long values that can include unsigned long integers, pointers to unsigned long integers, or pointers to characters.

IMPORTANT: The value in the first array element (at index zero) must give the number of elements in the array.

The *Table_Location* array can contain both long values and various kinds of pointers since it assumes that long values and pointers are both the same size.

Depending on the MIB definition, each element of the array can contain any one of the following type of attributes in the prescribed form:

Integer	Contained in the array as a pointer to an unsigned long.

TimeTicks	Contained in the array as a pointer to an unsigned long. The unit of time in this value must be NetWare operating system ticks.
short OCTET STRING	Contained in the array as a pointer to an unsigned long with the low order byte being the first octet of the string.
long OCTET STRING	Contains in the array as a pointer to an array of type char that contains the value. The value must be a NULL-terminated string. Any long OCTET STRING that can contain NULL must be specially handled by the support module of the object. If the string is writeable, the pointer must point to a buffer large enough to handle the maximum length value. Any long OCTET STRING that has no maximum length in the MIB is assumed to have a maximum of 256 characters, including the terminating NULL. If a truly unbound, writeable OCTET STRING must be supported by the object, the object support code must specially handle it.
OBJECT ID	Contained in the array as a pointer to an array of type unsigned long. The OBJECT ID is terminated by an element of zero. If the OBJECT ID is writeable, the pointer must point to a buffer large enough to handle a 256-element OBJECT ID. If a truly unbound, writeable OBJECT ID must be supported by the object, the object support code must specially handle it.

The format of this table and, consequently, the implementation of **SAI_DEFINE_TABLE** and its supporting functions can vary in different architectures that have different size pointers and unsigned long values.

The unit of time for TimeTicks can vary between environments.

See Also

SAI_DEFINE_GROUP, SAI_DEFINE_POINTER

Queue Management

Queue Management: Guides

Queue Management: General Guide

NetWare's queue management provides applications with generic queuing mechanisms.

Parent Topic:

Management Overview

Queue Management: Task Guide

NetWare's queue management provides applications with generic queuing mechanisms.

Managing Queue Jobs

Creating Queue Jobs

Related Topics:

Queue Management: Concepts

Queue Management: Functions

Queue Management: Structures

Parent Topic:

Queue Management: General Guide

Queue Management: Concept Guide

NetWare's queue management provides applications with generic queuing mechanisms.

Processing a Queue Job

Accessing the Queue Status Record

Queue Functions

Queue Job Functions

Queue Job Management Functions

Queue Management Functions

Queue Server Functions

Related Topics:

Queue Management: Tasks

Queue Management: Functions

Queue Management: Structures

Parent Topic:

Queue Management: General Guide

Managing a Queue Object

Queue Management provides full console-type control over the operation of the queue. You can create and delete queue objects and read and set queue status information for a queue.

Tasks

Creating a Queue Object

Deleting a Queue Object

Concepts

Queue Status Information

Parent Topic:

Queue Management: General Guide

Managing Queue Jobs

This topic details how to manage a queue job record.

Tasks

Creating Queue Jobs

Deleting Queue Jobs

Concepts

Modifiable Queue Job Information

Parent Topic:

Queue Management: General Guide

Queue Functions

This section provides lists of queue functions.

Queue Job Functions

Queue Job Management Functions

Queue Management Functions

Queue Server Functions

Parent Topic:

Queue Management: General Guide

Servicing a Queue Object

After the user and operator, the server is the third component in the queue management architecture. While the queue management system controls access to the queue and keeps the jobs in order, a queue server is responsible for removing jobs from the queue and processing them. Queue Management makes a few assumptions about the kind of services provided by a queue server.

Tasks

Attaching to a Queue Object

Aborting a Queue Job

Changing Queue Job Rights

Concepts

Processing a Queue Job

Accessing the Queue Status Record

Parent Topic:

Queue Management: General Guide

Queue Management: Tasks

Aborting a Queue Job

1. **If a queue server needs to interrupt a job in process, it calls NWAbortServicingQueueJob2.**

   This function allows the queue management system to handle the unfinished job according to the values of the job's control flags.

Parent Topic:

Servicing a Queue Object

Related Topics:

Processing a Queue Job

Changing Queue Job Rights

Attaching to a Queue Object

1. **To service a queue, a queue server application logs in to the NetWare® server and calls NWAttachQueueServerToQueue.**

   To attach to the queue successfully, a queue server must be listed in the queue's Q_SERVERS property.

2. **After attaching, a queue server calls NWServiceQueueJob2.**

   This function can include a job type identifying the type of job the server will accept. (A value of -1 includes all job types.)

Parent Topic:

Servicing a Queue Object

Changing Queue Job Rights

1. **While processing a job, a queue server can call NWChangeToClientRights2 to assume the access rights of the user who submitted the job.**

   This step is important if the job requires access to other files or bindery objects. By assuming the client's rights, a queue server can be sure the user is permitted to access the specified files.

2. **After the job is complete, a queue server calls NWRestoreQueueServerRights to have its rights restored.**

Parent Topic:

Servicing a Queue Object

Related Topics:

Processing a Queue Job

Aborting a Queue Job

Creating a Queue Object

1. **Call NWCreateQueue to create a queue object.**

   This function creates the queue object and assigns it a queue type, name, and directory path. The following queue types are defined by Novell:

0300h	Print Queue
0800h	Archive Queue
0A00h	Job Queue

   An 8-character hexadecimal representation of the queue's object ID is added to the directory path, and the resulting path is assigned to the queue's Q_DIRECTORY property. Typically, the directory SYS:SYSTEM is used as the directory path, although you can locate the queue somewhere else.

Parent Topic:

Managing a Queue Object

Related Topics:

Deleting a Queue Object

Queue Status Information

Deleting a Queue Object

1. **Call NWDestroyQueue to delete a queue object.**

   Deleting a queue deletes all associated data, including all jobs currently in the queue. Any active jobs associated with the queue are aborted.

Parent Topic:

Managing a Queue Object

Related Topics:

Creating a Queue Object

Queue Status Information

Creating Queue Jobs

1. **Call NWCreateQueueFile2 to create a queue job.**

   This function sets up a queue job record and opens a queue job file. Pass this function the destination queue and, optionally, the target server, target execution time, job type, and job control flags. You can also include a text job description. The client record area can be used to store any additional information required by your application.

   The queue job returns a job file handle to the queue file upon its creation. Use this handle to write data to the file.

2. **Call NWCloseFileAndStartQueueJob2 to close the queue job file and make the job available for processing.**

Parent Topic:

Managing Queue Jobs

Related Topics:

Deleting Queue Jobs

Modifiable Queue Job Information

Deleting Queue Jobs

1. **Call NWRemoveJobFromQueue2 to cancel a job and remove it from the queue.**

Parent Topic:

Managing Queue Jobs

Related Topics:

Creating Queue Jobs

Modifiable Queue Job Information

Queue Management: Concepts

Accessing the Queue Status Record

Queue management maintains a 64-byte status record for each queue to which a queue server is attached. Although this field is available for your application, queue management doesn't use it. Any object listed in either the queue's Q_USERS or Q_OPERATORS properties can read or modify this information. A pair of functions provide this service:

NWReadQueueServerCurrentStatus2

NWSetQueueServerCurrentStatus

Parent Topic:

Servicing a Queue Object

Modifiable Queue Job Information

Once in the queue, a job is identified by its queue job number. This number is part of the information returned in the queue job record when you submit the job. Use this number to read a the queue job record and to find the current size of the job file. You can modify the following information in the queue job record:

Target server ID number

Target execution time

Job type

User hold flag

Server restart flag

Server auto-start flag

Text job description in the client record area

Operator hold flag

Additionally, you can change the job's position in the queue or remove the job from the queue. To modify the operator hold flag, an application must have operator status. Operators can also receive a list of all jobs currently in the queue.

the queue.

Parent Topic:

Managing Queue Jobs

Related Topics:

Creating Queue Jobs

Deleting Queue Jobs

Processing a Queue Job

After receiving the request to service a job, queue management searches the queue for a job that is appropriate for the queue server. The following conditions must apply:

The job's target server ID number must match the queue server or be set to -1L.

The target execution time must have expired or be set to FFh.

The job's type must match the type specified by the server.

The operator hold, user hold, and entry open flags must all be clear, and the server ID number must be 0, indicating that the job isn't being serviced.

When an eligible job is found, the job's server station, server task, and server ID number are set to that of the queue server. The job file is opened for read/write access, and the updated queue job record is returned to the queue server. A queue server uses the job file handle to access the application-specific data associated with the job. When it has finished processing the job, a queue server calls **NWFinishServicingQueueJob2**. This is queue management's signal that the job has been processed.

Parent Topic:

Servicing a Queue Object

Related Topics:

Aborting a Queue Job

Changing Queue Job Rights

Queue Attributes

Queue attributes include Q_DIRECTORY, Q_OPERATOR, Q_SERVERS, and Q_USERS. The following table shows the configuration of these properties.

Table auto. The Queue Object Properties

Property Name	Property Flag	Property Security
Q_DIRECTORY	0x00 (item/static)	0x33
Q_OPERATOR	0x02 (set/static)	0x31
Q_SERVERS	0x02 (set/static)	0x31
Q_USERS (optional)	0x02 (set/static)	0x31

NOTE: Property security is based on the bindery access control scheme in which the upper nibble indicates write access and the lower nibble indicates read access. A value of 3 represents supervisor-only access and 1 represents any logged in object. For example, the Q_USERS property security is 0x31: only the supervisor can modify this property but any logged in user can read its value.

The Q_DIRECTORY property holds the path of the queue directory (for example, SYS:SYSTEM/055D0173). The queue object ID is used to name the subdirectory where the queue is found.

The Q_OPERATOR property holds the object IDs of all bindery objects assigned as queue operators. The Q_SERVERS property holds the object IDs of all servers that can access the queue. The Q_USERS property holds the object ID of all users who can access the queue.

Parent Topic:

Queue Management: General Guide

Queue Job Client Record Area

The client record area is a 152-byte field that can contain supplementary information exchanged between the queue client and the queue server. For NetWare® to process your print job correctly, you must enter the following into the clientRecordArea when the job is submitted:

```
typedef struct NWQPrintServerRec_t
    nuint8         versionNumber;
    nuint8         tabSize;
    nuint16        numberOfCopies;
    nuint16        printControlFlags;
    nuint16        maxLinesPerPage;
    nuint16        maxCharsPerLine;
    char           formName[13];
    nuint8         reserve[9];
    char           bannerNameField[13];
    char           bannerFileField[13];
    char           bannerFileName[14];
    char           directoryPath[80];
```

Parent Topic:

The Queue Job Record

Related Topics:

Queue Job Control Flags

Queue Job Control Flags

Job control flags affect the way the queue server processes a queue job. The flags define the following bits. (Bits 0, 1, and 2 must be 0.):

First Byte

Bit 3 = service auto-start
Bit 4 = service restart
Bit 5 = entry open
Bit 6 = user hold
Bit 7 = operator hold

service auto-start indicates how to handle the job should the client station lose its connection before submitting the job. If this flag is set, the job will be marked for processing should the client station lose its connection to the queue; otherwise the job is removed from the queue.

service restart indicates how to handle the job if the queue server fails during processing. If this flag is set, the job will be left in the queue for re-servicing.

entry open indicates whether the job is ready to be processed. When the job entry is first created, this flag is set to indicate the job hasn't been submitted yet. Once the client submits a job with this flag, the flag is cleared.

user hold indicates whether the job may be processed. If this flag is set, the job will not be processed until the flag is clear, but the job continues to advance toward the front of the queue. An operator or the client who submitted the job can modify this flag.

operator hold indicates whether the job may be processed. If the flag is set, the job will not be processed until the flag is clear, but the job continues to advance toward the front of the queue. Only operators can modify this flag.

Parent Topic:

The Queue Job Record

Related Topics:

Queue Job Client Record Area

Queue Job Functions

These functions let you create, abort, and close queue job files.

Function	Header	Comment
NWCloseFileAndAbortQueueJob2	nwqms.h	Closes a queue file and destroys it.
NWCloseFileAndStartQueueJob2	nwqms.h	Closes a queue file and marks the job ready for processing.
NWCreateQueueFile2	nwqms.h	Creates a queue file and returns a handle to it.

Parent Topic:

Queue Functions

Queue Job Management Functions

These functions let you manage queue jobs after they are submitted to the queue.

Function	Header	Comment
NWChangeQueueJobEntry2	nwqms.h	Changes the queue job information for the specified job.
NWChangeQueueJobPosition2	nwqms.h	Changes the position of a job in the specified queue.
NWGetQueueJobFileSize2	nwqms.h	Returns the size of the job file associated with a queue job.
NWGetQueueJobList2	nwqms.h	Returns a list of all entries in the specified queue.
NWReadQueueJobEntry2	nwqms.h	Reads the queue job information for the specified job.
NWRemoveJobFromQueue2	nwqms.h	Removes a job from the specified queue.

Parent Topic:

Queue Functions

The Queue Job Record

Within the queue, each job has a record that holds information for processing the job. The information identifies the client, the target server, the job, and the actual server:

Client station number

Client task

ClientID

Target server ID

Target execution time

Job entry time

Job number

Job type

Job position

Job control flags

Job file name

Job file handle

Servicing server station

Servicing server ID

Job description

Client record area

NWQueueJobStruct contains this data.

Parent Topic:

Queue Management: General Guide

Related Topics:

Queue Job Control Flags

Queue Job Client Record Area

Queue Management Functions

These functions let you create queues, delete queues, and manage queue status.

Function	Header	Comment
NWCreateQueue	nwqms.h	Creates a queue object in the bindery along with its associated queue properties.
NWDestroyQueue	nwqms.h	Deletes a queue object from the bindery and removes its queue directory. All entries and associated job files are also deleted.
NWGetPrinterQueueID	nwqms.h	Returns the object ID of the queue assigned to a specified LPT port.
NWReadQueueCurrentSt atus2	nwqms.h	Returns the current status of the specified queue.
NWSetQueueCurrentStat us2	nwqms.h	Allows the operator to modify a queue's queue status property. This property affects the submission of new jobs to the queue and the attachment of queue servers.

Parent Topic:

Queue Functions

Queue Management Introduction

NetWare's queue management provides applications with generic queuing mechanisms. A queue is simply an object that uses a directory on the NetWare® server where files (such as print jobs) can be stored and manipulated on a temporary basis. Although the primary use of queue management is processing print jobs, any file can be queued for any kind of processing.

Objects accessing the queue are defined as users, operators, or servers. A user can send a file to the queue and monitor its progress. An operator controls the queue's status and has access to any queue jobs currently found in the queue. A server removes jobs from the queue and performs the requested specialized services (such as printing).

Queue Management functions correspond with these three roles---user, operator, and server. User-oriented functions let users submit jobs to a queue from an application. Operator functions allow applications to manage and delete jobs from the queue. Server functions allow applications to remove jobs from a queue for processing.

Jobs submitted to a queue are copied into the queue directory on the network server. Print jobs are processed in the order they are submitted to the queue. Each file in the directory is given a number identifying its place relative to the other jobs in the queue. A hidden file in the queue directory keeps track of the sequence in which the jobs will be processed.

A job server checks each assigned queue at a specified interval, taking care of jobs on a first-in, first-out basis. As many as 25 job servers can be attached to a queue at the same time. Once a job is processed, its associated file is deleted from the queue directory.

For a description of structures and other data definitions relating to this topic, see Queue Management: Structures.

Parent Topic:

Queue Management: General Guide

Queue Objects

Like other bindery objects, a queue has a name, type, ID, object flag, and security flag. For example, a print queue might have the following object data:

Object name:	LASER_PRINTER	(user-supplied queue name)
Object type:	0x0300	(print queue)
Object ID:	055D0173	(assigned by NetWare®)
Object flag:	0x00	(static)
Object security:	0x31	(write = supervisor; read = logged)

Parent Topic:

Queue Management: General Guide

Queue Server Functions

These functions let queue servers attach to queues, open queue jobs, and assume client ID rights. They also provide access to the queue server status records associated with a queue.

Function	Header	Comment
NWAbortServicingQueueJob2	nwqms.h	Signals queue management that the

		server has discontinued servicing the specified job.
NWAttachQueueServerToQueue	nwqms.h	Attaches the local workstation as a queue server to the specified queue.
NWChangeToClientRights2	nwqms.h	Allows a queue server to change its current login identity to match the identity of the client whose job is being serviced.
NWDetachQueueServerFromQueue	nwqms.h	Removes the local workstation from the specified queue's list of active queue servers.
NWFinishServicingQueueJob2	nwqms.h	Signals queue management that the service has successfully serviced the job.
NWReadQueueServerCurrentStatus2	nwqms.h	Reads status information for a queue server. The information pertaining to the queue server status is specific to the application.
NWRestoreQueueServerRights	nwqms.h	Restores the queue server ID to a queue server. This function typically is called after **NWChangeToClientRights2** has changed the server's identity to the client ID.
NWServiceQueueJob2	nwqms.h	Returns the next available job in the specified queue.
NWSetQueueServerCurrentStatus	nwqms.h	Modifies the status information for a queue server.

Parent Topic:

Queue Functions

Queue Status Information

Queue status flags indicate a queue's current operational status. Below are the values associated with these flags.

First Byte

Bit 0 = QS_CANT_ADD_JOBS
Bit 1 = QS_SERVERS_CANT_ATTACH
Bit 2 = QS_CANT_SERVICE_JOBS

If QS_CANT_ADD_JOBS is set, no new jobs can be submitted to the queue.

If QS_SERVERS_CANT_ATTACH is set, no new servers can attach to the queue.

If QS_CANT_SERVICE_JOBS is set, the queue can't be serviced. A pair of functions access the queue status flags:

NWReadQueueCurrentStatus2 reads the flags.

NWSetQueueCurrentStatus2 modifies the flags.

Parent Topic:

Managing a Queue Object

Related Topics:

Creating a Queue Object

Deleting a Queue Object

Queue Management: Functions

NWAbortServicingQueueJob2

Signals to the queue management software a job previously accepted for service cannot be completed successfully

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: Queue Management System

Syntax

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

NWCCODE N_API NWAbortServicingQueueJob2 (
   NWCONN_HANDLE   conn,
   nuint32         QueueID,
   nuint32         JobNumber,
   NWFILE_HANDLE   fileHandle);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWAbortServicingQueueJob2
  (conn : NWCONN_HANDLE;
   QueueID : nuint32;
   JobNumber : nuint32;
   fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the job server connection handle.

QueueID

(IN) Specifies the bindery object ID for the queue in which the aborted job is located.

JobNumber

(IN) Specifies the job number of the job to abort (on servers previous to 3.11, the top 16 bits are ignored).

fileHandle

(IN) Specifies the file handle of the file associated with the aborted job.

Return Values

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

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_NO_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure, HARDWARE_FAILURE

Remarks

NWAbortServicingQueueJob2 is NetWare® 3.11+ compatible.

NWAbortServicingQueueJob2 closes the job file and resets the job server's access rights to their original (login) values. Only a queue server that has previously accepted a job for service can call **NWAbortServicingQueueJob2**.

An aborted job returns to its former position in the job queue if its QF_ENTRY_RESTART flag (bit 0x10 of *jobControlFlags* in QueueJobStruct) is set. For example, if a job is at the beginning of the queue before being called, it returns to the beginning of the queue after being aborted. Therefore, an aborted job could remain in the queue and be serviced and aborted again and again. A job should not be aborted because of an error in the job's format or requests. Instead, call **NWFinishServicingQueueJob2** to remove such a job from the queue.

Also, if a job attempts to access data without proper security clearance and is aborted, the job returns to the end of the queue. To remove such a job from the job queue, call **NWFinishServicingQueueJob2**.

A job should be aborted only if some temporary internal problem prevents it from completing. For example, a print job might be aborted if the printer has a paper jam. After the paper jam is corrected, the job server can service the job successfully.

NCP Calls

0x2222 66	File Close
0x2222 23 17	Get File Server Information
0x2222 23 115	Abort Servicing Queue Job (no 1000 user support)
0x2222 23 132	Abort Servicing Queue Job (3.11 and above)

See Also

NWChangeQueueJobEntry2, **NWCreateQueueFile2**, **NWFinishServicingQueueJob2**, **NWReadQueueJobEntry2**

NWAttachQueueServerToQueue

Attaches the calling station to the specified queue as a queue 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: Queue Management System

Syntax

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

NWCCODE N_API NWAttachQueueServerToQueue (
   NWCONN_HANDLE   conn,
   nuint32         queueID);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWAttachQueueServerToQueue
  (conn : NWCONN_HANDLE;
   queueID : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue being attached.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE

0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_NO_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89EA	NO_SUCH_MEMBER
0x89FC	NO_SUCH_OBJECT
0x89FF	Failure, HARDWARE_FAILURE

Remarks

A station must attach itself to a queue as a job server before it can service jobs from that queue. A queue can have as many as 25 job servers attached.

The workstation calling **NWAttachQueueServerToQueue** must be security equivalent to one of the objects listed in the queue's Q_SERVERS property.

NCP Calls

0x2222 23 111 Attach Queue Server To Queue

See Also

NWCreateQueue, **NWDetachQueueServerFromQueue**

NWChangeQueueJobEntry2

Changes the information about a job in a queue

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: Queue Management System

Syntax

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

NWCCODE N_API NWChangeQueueJobEntry2 (
   NWCONN_HANDLE            conn,
   nuint32                  queueID,
   NWQueueJobStruct N_FAR   job);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWChangeQueueJobEntry2
   (conn : NWCONN_HANDLE;
    queueID : nuint32;
    Var job : NWQueueJobStruct
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue.

job

(IN) Points to NWQueueJobStruct containing the new information about the job.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_NO_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure, HARDWARE_FAILURE

Remarks

Information supplied by the caller overwrites information already in the job record.

NWChangeQueueJobEntry2 can be used in conjunction with **NWReadQueueJobEntry2** to change a portion of the job's entry record. However, if the target entry is already being serviced, **NWChangeQueueJobEntry2** returns a servicing error and makes no changes to the job's entry record.

If the caller is an operator, QF_OPERATOR_HOLD can be reset to a value supplied by the caller.

An operator or the job creator can call **NWChangeQueueJobEntry2**.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 109	Change Queue Job Entry (no 1000 user support)
0x2222 23 123	Change Queue Job Entry (3.11 and above)

See Also

NWChangeQueueJobPosition2, **NWGetQueueJobList2**, **NWReadQueueJobEntry2**

NWChangeQueueJobPosition2

Changes a job's position in a queue
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: Queue Management System

Syntax

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

NWCCODE N_API NWChangeQueueJobPosition2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber,
   nuint32         newJobPos);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWChangeQueueJobPosition2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32;
   newJobPos : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the affected queue.

jobNumber

(IN) Specifies the job number of the job being repositioned.

newJobPos

(IN) Specifies the job's new position.

Return Values

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

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89FE	BINDERY_LOCKED
0x89FF	Failure

Remarks

newJobPos ranges from 1 to 250. Position 1 is the first position in the queue and position 250 is the last position in a full queue. If a specified position number places the job beyond the current end of the queue, the job moves to the end of the current queue.

When a job is moved in the queue, the positions of all job entries are updated to reflect the change. Changing the position of a job being serviced has no effect on the service of that job. Also, note that moving a job in the queue does not change *jobNumber*, only its position.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 110 Change Queue Job Position (no 1000 user support)
0x2222 23 130 Change Queue Job Priority (3.11 or later)

See Also

NWChangeQueueJobEntry2, **NWGetQueueJobList2**, **NWReadQueueJobEntry2**, **NWRemoveJobFromQueue2**

NWChangeToClientRights2

Allows a queue server to change its current login identity to match the identity of the client for which it is acting

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: Queue Management System

Syntax

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

NWCCODE N_API NWChangeToClientRights2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWChangeToClientRights2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the queue server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue.

jobNumber

(IN) Specifies the job's number.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_NO_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure, HARDWARE_FAILURE

Remarks

The queue server's login user ID and associated security equivalence list are replaced by the ID and security equivalence list of the user who placed the job in the queue.

NWChangeToClientRights2 does not change any path mappings the queue server may have on the job server. However, all access rights to those directories are recalculated to conform to the rights of the queue client. Files opened before calling **NWChangeToClientRights2** continue to be accessible with the server's rights. Files opened after calling **NWChangeToClientRights2** are accessible only with the client's rights.

The job server creates path mappings needed to carry out the client's requests after calling **NWChangeToClientRights2**.

NWRestoreQueueServerRights reverses the effects of **NWChangeToClientsRights2**. The server's rights are automatically reset if the server calls **NWFinishServicingQueueJob2** or **NWAbortServicingQueueJob2**.

Only a queue server that has previously accepted a job for service can call **NWChangeToClientRights2**.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 116	Change To Client Rights (no 1000 user support)

0x2222 23 133   Change To Client Rights (3.11 and above)

See Also

NWAbortServicingQueueJob2, **NWFinishServicingQueueJob2**, **NWRestoreQueueServerRights**

NWCloseFileAndAbortQueueJob2

Allows the workstation to close a queue job and abort it
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: Queue Management System

Syntax

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

NWCCODE N_API NWCloseFileAndAbortQueueJob2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber,
   NWFILE_HANDLE   fileHandle);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWCloseFileAndAbortQueueJob2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32;
   fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the queue server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue.

jobNumber

(IN) Specifies the job entry number of the job whose service is being aborted.

fileHandle

(IN) Specifies the aborted job's file handle returned by **NWCreateQueueFile2**.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_QUEUE_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	HARDWARE_FAILURE

Remarks

jobNumber contains the job number returned by the Queue Management System when the job was originally entered in the queue. The file associated with the job number is closed, and the job is deleted from the queue.

Only the workstation creating the queue job or a queue operator can call **NWCloseFileAndAbortQueueJob2**.

NCP Calls

0x2222 66 File Close

0x2222 23 17 Get File Server Information

0x2222 23 106 Remove Job From Queue (no 1000 user support)

0x2222 23 128 Remove Job From Queue (3.11 and above)

See Also

NWCloseFileAndStartQueueJob2, **NWCreateQueueFile2**, **NWRemoveJobFromQueue2**

NWCloseFileAndStartQueueJob2

Closes a queue file and marks it ready for execution
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: Queue Management System

Syntax

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

NWCCODE N_API NWCloseFileAndStartQueueJob2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber,
   NWFILE_HANDLE   fileHandle);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWCloseFileAndStartQueueJob2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32;
   fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the queue server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue in which the specified job was placed.

jobNumber

(IN) Specifies the job number of the job to be serviced.

fileHandle

(IN) Specifies the file handle of the job to be executed (returned by **NWCreateQueueFile2**).

Return Values

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

0x0000	SUCCESSFUL
0x0006	Queue file specified by *fileHandle* does not exist.
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_QUEUE_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure, HARDWARE_FAILURE

Remarks

jobNumber contains the job number returned by Queue Management System when the job was originally entered in the queue.

NWCloseFileAndStartQueueJob2 closes the file associated with the job and resets QF_ENTRY_OPEN in *jobControlFlags*.

jobControlFlags bit definitions follow:

Bit	Flag Name	Description
0x08	Auto Start	Specifies how to handle the job should the client station lose its connection before submitting the job. If this bit is set, the job is serviced after a queue server connection is broken, even if the client has not cleared the Entry Open bit. If the bit is cleared when a server connection is broken, Queue Management System removes the job from

		the queue.
0x10	Service Restart	Specifies how to handle the job if the queue server fails during processing. The job remains in the queue (in its current position) when a queue server fails. If this bit is cleared, Queue Management System removes the job from the queue when a server fails.
0x20	Entry Open	Specifies whether the job is ready to be processed. When the job entry is first created, this flag is set to indicate the job has not been submitted yet. When the client submits the job, this flag is cleared. **NWCloseFileAndStartQueueJob2** clears this bit (marking the job ready for service) if the User Hold and Operator Hold bits are cleared.
0x40	User Hold	Specifies whether the job may be processed. If the flag is set, the job will not be processed until the flag is clear, but the job continues to advance toward the front of the queue. An operator or the client who submitted the job can modify this flag.
0x80	Operator Hold	Specifies whether the job may be processed. If the flag is set, the job will not be processed until the flag is clear, but the job continues to advance toward the front of the queue. Only operators can modify this flag.

The specified job is ready for execution when (1) **NWCloseFileAndStartQueueJob2** finishes, (2) QF_USER_HOLD and QF_OPERATOR_HOLD are both cleared, and (3) *targetExecutionTime* in NWQueueJobStruct either has not been specified or has elapsed.

Only the workstation that created the job can call **NWCloseFileAndStartQueueJob2**.

NCP Calls

0x2222 23 17   Get File Server Information

0x2222 23 105   Close File And Start Queue Job (no 1000 user support)

0x2222 23 127   Close File And Start Queue Job (3.11 or later)

0x2222 66   File Close

See Also

NWCloseFileAndAbortQueueJob2, **NWCreateQueueFile2**, **NWRemoveJobFromQueue2**

NWCreateQueue

Creates a new queue and its associated Q_DIRECTORY property in the Bindery and file system of the specified 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: Queue Management System

Syntax

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

NWCCODE N_API NWCreateQueue (
   NWCONN_HANDLE   conn,
   pnstr8          queueName,
   nuint16         queueType,
   nuint8          dirPath,
   pnstr8          path,
   pnuint32        queueID);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWCreateQueue
  (conn : NWCONN_HANDLE;
   queueName : pnstr8;
   queueType : nuint16;
   dirPath : nuint8;
   path : pnstr8;
   queueID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueName

(IN) Points to the name of the queue to be created (48 characters).

queueType

(IN) Specifies the number indicating the Bindery object ID type for the new queue.

dirPath

(IN) Specifies the NetWare directory handle pointing to the directory in which the queue's property is to be created (0 if the queue subdirectory name will be stored in Q_DIRECTORY).

path

(IN) Points to the absolute path or a path relative to the NetWare directory handle that will contain the queue files (stored in Q_DIRECTORY).

queueID

(OUT) Points to the new queue ID.

Return Values

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

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_QUEUE_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89EE	OBJECT_ALREADY_EXISTS
0x89FF	Failure

Remarks

Q_DIRECTORY is determined by combining *dirPath* and *path*. **NWCreateQueue** then creates a new subdirectory in this directory. The subdirectory's name is the 8-character ASCII hexadecimal representation of the new queue’s Bindery object ID. Queue Management System uses this directory to store queue files until they are serviced.

Next, **NWCreateQueue** creates the following group properties:

```
Q_SERVERS
Q_OPERATORS
Q_USERS
```

Only SUPERVISOR or a Bindery object that is security equivalent to SUPERVISOR can create a queue.

dirPath of 0 and SYS:SYSTEM is standard usage.

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 23 100   Create Queue

See Also

NWDestroyQueue

NWCreateQueueFile2

Enters a new job on the queue and creates a job 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: Queue Management System

Syntax

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

NWCCODE N_API NWCreateQueueFile2 (
   NWCONN_HANDLE            conn,
   nuint32                  queueID,
   NWQueueJobStruct N_FAR  *job,
   NWFILE_HANDLE N_FAR    *fileHandle);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWCreateQueueFile2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   Var job : NWQueueJobStruct;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery's object ID for the queue.

job

(IN/OUT) Points to NWQueueJobStruct, which stores the information about the job.

fileHandle

(OUT) Points to the file handle.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_QUEUE_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FC	NO_SUCH_OBJECT
0x89FF	Failure

Remarks

After creating a job file, **NWCreateQueueFile2** attaches it to a special file opened at the workstation. The special file handle is returned in *fileHandle.*

The requesting workstation provides the Bindery ID of the queue the job should be appended to, as well as the address of the entire 256-byte job record (NWQueueJobStruct).

NWCreateQueueFile2 fills in the job record and returns the record (minus *jobDescription* and *clientRecordArea*) to the requesting workstation. After **NWCreateQueueFile2** is called, all fields are initialized.

NWCreateQueueFile2 can be used in conjunction with **NWReadQueueJobEntry2** to change a portion of the job's entry record. However, if the target entry is already being serviced, **NWChangeQueueJobEntry2** returns Q_SERVICING and makes no changes to the job's entry record.

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 23 104   Create Queue Job And File (OS/2 only; no 1000 user support)

0x2222 23 121   Create Queue Job And File (OS/2 only 3.11 or later)

0x2222 66   File Close

See Also

NWChangeQueueJobEntry2, **NWCloseFileAndStartQueueJob2**, **NWRemoveJobFromQueue2**, **NWReadQueueJobEntry2**

NWDestroyQueue

Deletes the specified queue
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: Queue Management System

Syntax

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

NWCCODE N_API NWDestroyQueue (
   NWCONN_HANDLE   conn,
   nuint32         queueID);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWDestroyQueue
  (conn : NWCONN_HANDLE;
   queueID : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue to be deleted.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE

0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_QUEUE_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

All active jobs are aborted, all servers are detached from the queue, and all jobs in the queue are deleted, along with their associated files. The queue object and its associated properties are also removed from the Bindery and the queue's subdirectory is deleted.

Only SUPERVISOR or a Bindery object that is security equivalent to SUPERVISOR can destroy a queue.

NCP Calls

0x2222 23 101 Destroy Queue

NWDetachQueueServerFromQueue

Removes the requesting station from the queue's list of active queue servers
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: Queue Management System

Syntax

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

NWCCODE N_API NWDetachQueueServerFromQueue (
   NWCONN_HANDLE   conn,
   nuint32         queueID);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWDetachQueueServerFromQueue
   (conn : NWCONN_HANDLE;
    queueID : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the queue's bindery object ID from which the calling station is being detached.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE

0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_QUEUE_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

If the requesting workstation is servicing a job, the service is automatically aborted.

Only a workstation previously attached to the queue as a queue server can call **NWDetachQueueServerFromQueue**.

NCP Calls

0x2222 23 112 Detach Queue Server From Queue

See Also

NWAttachQueueServerToQueue,
NWReadQueueServerCurrentStatus2,
NWSetQueueServerCurrentStatus

NWFinishServicingQueueJob2

Allows a queue server to signal Queue Management System it has serviced a job successfully

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: Queue Management System

Syntax

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

NWCCODE N_API NWFinishServicingQueueJob2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber,
   NWFILE_HANDLE   fileHandle);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWFinishServicingQueueJob2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32;
   fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue containing the job being finished.

jobNumber

(IN) Specifies the job's number being finished.

fileHandle

(IN) Specifies the file handle pointing to the file associated with the queue job.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

The job entry is destroyed, and the job file is closed and deleted. The calling queue server's access rights to the queue server are restored to their original (login) values.

Only a queue server accepting a job to service can call **NWFinishServicingQueueJob2**.

NCP Calls

0x2222 23 17 Get File Server Information

0x2222 23 114 Finish Servicing Queue Job (no 1000 user support)

0x2222 23 131 Finish Servicing Queue Job (3.11 or later)

0x2222 66 File Close

See Also

NWAbortServicingQueueJob2, **NWChangeToClientRights2**, **NWServiceQueueJob2**

NWGetPrinterQueueID

Returns the object ID of the queue servicing the specified printer

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: Queue Management System

Syntax

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

NWCCODE N_API NWGetPrinterQueueID (
   NWCONN_HANDLE   conn,
   nuint16         printerNum,
   pnuint32        queueID);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWGetPrinterQueueID
  (conn : NWCONN_HANDLE;
   printerNum : nuint16;
   queueID : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

printerNum

(IN) Specifies the local LPT device number whose output is assigned to the queue.

queueID

(OUT) Points to the bindery object ID of the print queue.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89FF	BAD_PRINTER_ERROR

Remarks

printerNum can have one of the following values which are assigned to the server not the workstation:

```
1 = LPT1
2 = LPT2
3 = LPT
```

For example, with LPT1 on queue name RIGHT_BRAIN, type on the server console screen:

```
spool LPT1 RIGHT_BRAIN
```

NCP Calls

0x2222 17 10   Get Printer's Queue

NWGetQueueJobFileSize2

Returns the file size of the file associated with a queue entry

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: Queue Management System

Syntax

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

NWCCODE N_API NWGetQueueJobFileSize2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber,
   pnuint32        fileSize);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWGetQueueJobFileSize2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32;
   fileSize : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue with which the job is associated.

jobNumber

(IN) Specifies the job number for which the information will be obtained.

fileSize

(OUT) Points to the queue job’s file size.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

If the file associated with the queue entry is still open, the file size returned does not necessarily reflect the file's final size.

NCP Calls

0x2222 23 17	Get File Server Information
0x2222 23 120	Get Queue Job File Size (no 1000 user support)
0x2222 23 135	Get Queue Job Size (3.11 or later)

NWGetQueueJobList2

Returns a list of all the jobs currently in a queue
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: Queue Management System

Syntax

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

NWCCODE N_API NWGetQueueJobList2 (
   NWCONN_HANDLE              conn,
   nuint32                    queueID,
   nuint32                    queueStartPos,
   QueueJobListReply N_FAR   *job);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWGetQueueJobList2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   queueStartPos : nuint32;
   Var job : QueueJobListReply
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue whose job list is being reported.

queueStartPos

(IN) Specifies the job number from which to start reading jobs.

job

(OUT) Points to QueueJobListReply containing the job numbers of all the jobs in the queue.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

When used in conjunction with **NWReadQueueJobEntry2**, **NWGetQueueJobList2** allows an application to retrieve information about all the jobs in a given queue. Because the Queue Management System environment is multi-threaded, the positioning, number, and type of jobs in the queue can change between consecutive calls.

NWGetQueueJobList2 allows a workstation to determine how many jobs are in the queue at a particular instant and the job number of each. If a subsequent call to read information about a job in the queue fails with NO_Q_JOB, the requesting workstation can assume either the job was deleted from the queue or its service was completed.

NWGetQueueJobList2 will only read in 125 jobs at a time even though the job array is defined as 250. If you wish to read a job that is above 125, set *queueStartPos* to 126 and allow **NWGetQueueJobList2** to read job numbers 126 through 251.

The workstation calling **NWGetQueueJobList2** must be security equivalent to one of the objects listed in the queue’s Q_USERS or Q_OPERATORS group properties.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 107 Get Queue Job List (no 1000 user support)
0x2222 23 129 Get Queue Job List (3.11 and above)

See Also

NWChangeQueueJobEntry2, **NWChangeQueueJobPosition2**, **NWReadQueueJobEntry2**

NWReadQueueCurrentStatus2

Reads the current status of the specified queue

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: Queue Management System

Syntax

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

NWCCODE N_API NWReadQueueCurrentStatus2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   pnuint32        queueStatus,
   pnuint32        numberOfJobs,
   pnuint32        numberOfServers,
   pnuint32        serverIDlist,
   pnuint32        serverConnList);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWReadQueueCurrentStatus2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   queueStatus : pnuint32;
   numberOfJobs : pnuint32;
   numberOfServers : pnuint32;
   serverIDlist : pnuint32;
   serverConnList : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue for which the status is being obtained.

queueStatus

(OUT) Points to the status of the specified queue (optional).

numberOfJobs

(OUT) Points to the number of jobs currently in the queue, 0-250 (optional).

numberOfServers

(OUT) Points to the number of queue servers currently attached to service the queue, 0-25 (optional).

serverIDlist

(OUT) Points to an array of server IDs associated with*numberOfServers* (25 nuint32s, optional).

serverConnList

(OUT) Points to an array of station numbers corresponding to the servers returned by *serverIDlist* (25 nuint32s, optional).

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

queueStatus

as follows:

C Value	Pascal Value	Value Name
0x01	$01	QS_CANT_ADD_JOBS
0x02	$02	QS_SERVERS_CANT_ATTACH
0x04	$04	QS_CANT_SERVICE_JOBS

serverIDList and *serverConnList* identify queue servers currently servicing the queue by object ID number and current workstation attachment.

NOTE: Workstations calling **NWReadQueueCurrentStatus2** must be security equivalent to one of the objects listed in the queue's Q_USERS, Q_OPERATORS, or Q_SERVERS properties.

NCP Calls

0x2222 23 17 Get File Server Information

0x2222 23 102 Read Queue Current Status (no 1000 user support)

0x2222 23 125 Read Queue Current Status (3.11 or later)

See Also

NWAttachQueueServerToQueue, **NWDetachQueueServerFromQueue**, **NWReadQueueServerCurrentStatus2**, **NWSetQueueCurrentStatus2**, **NWSetQueueServerCurrentStatus**

NWReadQueueJobEntry2

Allows an application to retrieve information about a job from a queue
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: Queue Management System

Syntax

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

NWCCODE N_API NWReadQueueJobEntry2 (
   NWCONN_HANDLE             conn,
   nuint32                   queueID,
   nuint32                   jobNumber,
   NWQueueJobStruct N_FAR  *job);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWReadQueueJobEntry2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   jobNumber : nuint32;
   Var job : NWQueueJobStruct
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue associated with the queue job being read.

jobNumber

(IN) Specifies the job number being read.

job

(OUT) Points to NWQueueJobStruct returning the queue job information.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

The job's full 256-byte record is returned, as defined in nwqms.h.

NWReadQueueJobEntry2 fills in the job record and returns the record to the requesting workstation.

Workstations calling **NWReadQueueJobEntry2** must be security equivalent to one of the objects listed in the queue's Q_USER or Q_OPERATORS group properties.

NCP Calls

0x2222 23 108	Read Queue Job Entry (no 1000 user support)
0x2222 23 108	Read Queue Job Entry (no 1000 user support)
0x2222 23 122	Read Queue Job Entry (3.11 or later)

See Also

NWChangeQueueJobEntry2, **NWChangeQueueJobPosition2**, **NWCreateQueueFile2**, **NWGetQueueJobList2**

NWReadQueueServerCurrentStatus2

Allows a station to read the current status of a queue 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: Queue Management System

Syntax

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

NWCCODE N_API NWReadQueueServerCurrentStatus2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         serverID,
   nuint32         serverConn,
   nptr            statusRec);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWReadQueueServerCurrentStatus2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   serverID : nuint32;
   serverConn : nuint32;
   statusRec : nptr
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue being affected.

serverID

(IN) Specifies the bindery object ID of the queue server whose current status is being read.

serverConn

(IN) Specifies the connection number of the queue server being read.

statusRec

(OUT) Points to a buffer containing the status of the specified queue server (64 bytes).

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

Queue Management System maintains a 64-byte status record for each queue server attached to a queue.

Queue Management System does not interpret the contents of the status record. The record contains information important to the calling application. It is recommended that the first 4 bytes of this record contain an estimated cost for the given server to complete a standard job (in hours, dollars, or whatever you define), to both indicate and standardize the function.

Workstations calling **NWReadQueueServerCurrentStatus2** must be security equivalent to one of the objects listed in the queue's Q_USER or Q_OPERATORS properties.

NCP Calls

0x222 23 17   Get Server Info
0x2222 23 118   Read Queue Server Current Status (no 1000 user support)
0x2222 23 134   Read Queue Server Current Status

See Also

NWSetQueueServerCurrentStatus

NWRemoveJobFromQueue2

Allows the workstation to remove a job from a queue

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: Queue Management System

Syntax

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

NWCCODE N_API NWRemoveJobFromQueue2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         jobNumber);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWRemoveJobFromQueue2
   (conn : NWCONN_HANDLE;
    queueID : nuint32;
    jobNumber : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue where the job to be removed is located.

jobNumber

(IN) Specifies the job number being removed.

Return Values

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

0x0000	SUCCESSFUL
0x8988	INVALID_FILE_HANDLE
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

jobNumber contains the job number returned by Queue Management System when the job was created. *jobNumber* can also be obtained by calling **NWGetQueueJobList2**.

The specified job is removed from the queue, and the job file is closed and deleted. If the job is being serviced, the service is aborted. Further I/O requests made to the job's queue file return INVALID_FILE_HANDLE.

Both the job's creator and an operator can call **NWRemoveJobFromQueue2**.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 106 Remove Job From Queue (no 1000 user support)
0x2222 23 128 Remove Job From Queue (3.11 or later)

See Also

NWChangeQueueJobEntry2, **NWChangeQueueJobPosition2**, **NWCreateQueueFile2**, **NWGetQueueJobList2**, **NWReadQueueJobEntry2**

NWRestoreQueueServerRights

Allows a queue server to restore its own identity after it has assumed its client's identity by calling **NWChangeToClientRights2**

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: Queue Management System

Syntax

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

NWCCODE N_API NWRestoreQueueServerRights (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwqms.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER

0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

After calling **NWRestoreQueueServerRights**, the queue server's login user ID and associated security equivalence list are restored to their original values.

NWRestoreQueueServerRights does not change any of the path mappings (directory bases) held by the queue server. However, access rights to those directories are adjusted to reflect the queue server's rights in those directories.

If the queue server has changed some of its path mappings as part of its efforts to service the queue job, the queue server must restore those directory bases.

Files opened using the client's rights before calling **NWRestoreQueueServerRights** continue to be accessible with the client's rights. Files opened after calling **NWRestoreQueueServerRights** are accessible only with rights of the queue server.

NWRestoreQueueServerRights does not need to be called if either **NWAbortServicingQueueJob2** or **NWFinishServicingQueueJob2** were called.

Only queue servers previously changing their identity by calling **NWChangeToClientRights2** can call **NWRestoreQueueServerRights**.

NCP Calls

0x2222 23 117 Restore Queue Server Rights

See Also

NWAbortServicingQueueJob2, **NWChangeToClientRights2**, **NWFinishServicingQueueJob2**

NWServiceQueueJob2

Allows a queue server to select a new job for servicing
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: Queue Management System

Syntax

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

NWCCODE N_API NWServiceQueueJob2 (
   NWCONN_HANDLE            conn,
   nuint32                  queueID,
   nuint16                  targetJobType,
   NWQueueJobStruct N_FAR  *job,
   NWFILE_HANDLE N_FAR     *fileHandle);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWServiceQueueJob2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   targetJobType : nuint16;
   Var job : NWQueueJobStruct;
   Var fileHandle : NWFILE_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue whose jobs are being serviced.

targetJobType

(IN) Specifies the job type to be serviced.

job

(OUT) Points to NWQueueJobStruct containing the job record of the next available job returned.

fileHandle

(OUT) Points to the file handle of the file associated with the job to be serviced.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

The requesting workstation must have previously established itself as a queue server for the target queue.

NWServiceQueueJob2 will fail if there are not enough directory handles available.

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 23 108	Read Queue Job Entry (no 1000 user support)
0x2222 23 113	Serivce Queue Job (no 1000 user support)
0x2222 23 120	Get Queue Job File Size
0x2222 23 122	Read Queue Job Entry (3.11 or later)
0x2222 23 124	Service Queue Job
0x2222 23 125	Service Queue Job (3.11 or later)
0x2222 23 135	Get Queue Job File Size
0x2222 66	File Close

See Also

NWAbortServicingQueueJob2, **NWAttachQueueServerToQueue**, **NWCreateQueueFile2**, **NWFinishServicingQueueJob2**, **NWReadQueueJobEntry2**

NWSetQueueCurrentStatus2

Allows the operator to control the addition of jobs and servers to the queue
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: Queue Management System

Syntax

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

NWCCODE N_API NWSetQueueCurrentStatus2 (
   NWCONN_HANDLE   conn,
   nuint32         queueID,
   nuint32         queueStatus);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWSetQueueCurrentStatus2
  (conn : NWCONN_HANDLE;
   queueID : nuint32;
   queueStatus : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue whose status is being updated.

queueStatus

(IN) Specifies the control byte determining the new queue status.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FC	NO_SUCH_OBJECT
0x89FF	Failure

Remarks

NWSetQueueCurrentStatus2 modifies the bits in*queueStatus*. The following queue status values are defined for **NWSetQueueCurrentStatus2**:

C Value	**Pascal Value**	**Value Name**
0x01	$01	QS_CANT_ADD_JOBS
0x03	$02	QS_SERVERS_CANT_ATTACH
0x04	$04	QS_CANT_SERVICE_JOBS

The workstation calling **NWSetQueueCurrentStatus2** must be logged in as one of the objects listed in Q_OPERATORS.

NCP Calls

0x2222 23 17 Get File Server Information
0x2222 23 103 Set Queue Current Status (no 1000 user support)
0x2222 23 126 Set Queue Current Status (3.11 or later)

See Also

NWAttachQueueServerToQueue, **NWDetachQueueServerFromQueue**, **NWReadQueueCurrentStatus2**

NWSetQueueServerCurrentStatus

Updates the Queue Management System copy of a servers status record
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: Queue Management System

Syntax

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

NWCCODE N_API NWSetQueueServerCurrentStatus (
   NWCONN_HANDLE    conn,
   nuint32          queueID,
   nptr             statusRec);
```

Pascal Syntax

```
#include <nwqms.inc>

Function NWSetQueueServerCurrentStatus
   (conn : NWCONN_HANDLE;
    queueID : nuint32;
    statusRec : nptr
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

queueID

(IN) Specifies the bindery object ID of the queue to which the specified queue server is attached.

statusRec

(IN) Points to the 64-byte buffer containing the new status record of the queue server.

Return Values

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

0x0000	SUCCESSFUL
0x8999	DIRECTORY_FULL
0x89D0	ERR_Q_IO_FAILURE
0x89D1	ERR_NO_QUEUE
0x89D2	ERR_NO_Q_SERVER
0x89D3	ERR_NO_Q_RIGHTS
0x89D4	ERR_Q_FULL
0x89D5	ERR_NO_Q_JOB
0x89D6	ERR_NO_Q_JOB_RIGHTS
0x89D7	ERR_Q_IN_SERVICE
0x89D8	ERR_Q_NOT_ACTIVE
0x89D9	ERR_Q_STN_NOT_SERVER
0x89DA	ERR_Q_HALTED
0x89DB	ERR_Q_MAX_SERVERS
0x89FF	Failure

Remarks

Queue Management System does not interpret the contents of the status record. The record contains information important to the calling application only. It is recommended the first four bytes of this record contain an estimated cost (in hours, dollars, or whatever is defined) for the given server to complete a standard job to both indicate and standardize the function.

Only workstations previously attached to the queue as queue servers can call **NWSetQueueServerCurrentStatus**.

NCP Calls

0x2222 23 119 Set Queue Server Current Status

See Also

NWReadQueueServerCurrentStatus2

Queue Management: Structures

NWQueueJobStruct

Holds the information found in the queue job record
Service: Queue Management
Defined In: nwqms.h

Structure

```
typedef struct
{
   nuint32   clientStation;
   nuint32   clientTask;
   nuint32   clientID;
   nuint32   targetServerID;
   nuint8    targetExecutionTime[6];
   nuint8    jobEntryTime[6];
   nuint32   jobNumber;
   nuint16   jobType;
   nuint16   jobPosition;
   nuint16   jobControlFlags;
   nuint8    jobFileName[14];
   nuint32   jobFileHandle[6];
   nuint32   servicingServerStation;
   nuint32   servicingServerTask;
   nuint32   servicingServerID;
   nuint8    jobDescription[50];
   nuint8    clientRecordArea[152];
} NWQueueJobStruct;
```

Pascal Structure

```
Defined in nwqms.inc

 NWQueueJobStruct = Record
    clientStation : nuint32;
    clientTask : nuint32;
    clientID : nuint32;
    targetServerID : nuint32;
    targetExecutionTime : Array[0..5] Of nuint8;
    jobEntryTime : Array[0..5] Of nuint8;
    jobNumber : nuint32;
    jobType : nuint16;
    jobPosition : nuint16;
    jobControlFlags : nuint16;
    jobFileName : Array[0..13] Of nuint8;
    jobFileHandle : nuint32;
    servicingServerStation : nuint32;
    servicingServerTask : nuint32;
    servicingServerID : nuint32;
    jobDescription : Array[0..49] Of nuint8;
    clientRecordArea : Array[0..151] Of nuint8
  End;
```

Fields

clientStation

Indicates the station number of the client submitting the job to the queue.

clientTask

Indicates the task number the station was performing when it placed the job in the queue.

clientID

Indicates the object ID of the person submitting the job.

targetServerID

Indicates the server ID of the queue server to service the job. If set to 0xFFFFFFFF, any queue server can service the job. If the specified queue server is not attached to the queue, Queue Management System Service removes the job from the queue.

targetExecutionTime

Indicates the earliest time the job can be serviced. The time is in the format: year, month, day, hour, minute, second (YYMMDDHHMMSS). If this field is set to 0xFFFFFFFFFFFF, the job is serviced at the first opportunity.

jobEntryTime

Indicates the time the client submitted the job to the queue.

jobNumber

Indicates a number used to identify the job.

jobType

Indicates a number identifying the type of job entry. A queue server can request specific job types from a queue. This value cannot be -1.

jobPosition

Indicates the job's position in the queue. The first job is assigned position 1, the next is assigned position 2, and so on. As jobs are removed from the queue, this number changes to reflect the updated position of the queue job.

jobControlFlags

Indicates flag bits indicating the status of the job. Bits 0, 1, and 2 must be 0.

jobFileName

Indicates the name of the job file (in DOS 8.3 format) created by Queue Management System. The name of the file is in the form Q$XXXX.YYY.

jobFileHandle

Indicates the handle of the job file created by Queue Management System Service. The name of the file is in the form Q$XXXX.YYY. It is normally not used by the client. Queue Management System provides the client with a file handle specific to the local file system (DOS or OS/2) for accessing the job.

servicingServerStation

Indicates the station number of the queue server servicing the job. If the job isn't being processed, this value is undefined.

servicingServerTask

Indicates the task number of the queue server servicing the job. If the job isn't being processed, this value is undefined.

servicingServerID

Indicates the server. If the job isn't being processed, this value is undefined.

jobDescription

Indicates the NULL-terminated ASCII text description of the content or purpose of a job. Queue Management System Service displays this text as part of the job description when users or operators examine a queue.

clientRecordArea

Indicates supplementary information that can be exchanged between the queue client and the queue server. The format for this information is defined by your application.

NWReplyJobStruct

Service: Queue Management
Defined In: nwqms.h

Structure

```
typedef struct
{
   nuint32   clientStation;
   nuint32   clientTask;
   nuint32   clientID;
   nuint32   targetServerID;
   nuint8    targetExecutionTime[6];
   nuint8    jobEntryTime[6];
   nuint32   jobNumber;
   nuint16   jobType;
   nuint16   jobPosition;
   nuint16   jobControlFlags;
   nuint8    jobFileName[14];
   nuint32   jobFileHandle;
   nuint32   servicingServerStation;
   nuint32   servicingServerTask;
   nuint32   servicingServerID;
} NWReplyJobStruct;
```

Pascal Structure

```
Defined in nwqms.inc

NWReplyJobStruct = Record
    clientStation : nuint32;
    clientTask : nuint32;
    clientID : nuint32;
    targetServerID : nuint32;
    targetExecutionTime : Array[0..5] Of nuint8;
    jobEntryTime : Array[0..5] Of nuint8;
    jobNumber : nuint32;
    jobType : nuint16;
    jobPosition : nuint16;
    jobControlFlags : nuint16;
    jobFileName : Array[0..13] Of nuint8;
    jobFileHandle : nuint32;
    servicingServerStation : nuint32;
    servicingServerTask : nuint32;
    servicingServerID : nuint32
  End;
```

Fields

Fields

clientStation

Indicates the client submitting the job to the queue.

clientTask

Indicates the task number the station was performing when it placed the job in the queue.

clientID

Indicates the object ID of the person submitting the job.

targetServerID

Indicates the server ID of the queue server servicing the job. If this field is set to 0xFFFFFFFF, any queue server can service the job. If the specified queue server is not attached to the queue, Queue Management System Service removes the job from the queue.

targetExecutionTime

Indicates the earliest time the job can be serviced. The time is in the format: year, month, day, hour, minute, second (YYMMDDHHMMSS). If this field is set to 0xFFFFFFFFFFFF, the job is serviced at the first opportunity.

jobEntryTime

Indicates the time the client submitted the job to the queue.

jobNumber

Indicates a number used to identify the job.

jobType

Indicates the number identifying the type of job entry. A queue server can request specific job types from a queue. This value cannot be -1.

jobPosition

Indicates the job’s position in the queue. The first job is assigned position 1, the next is assigned position 2, and so on. As jobs are removed from the queue, this number changes to reflect the updated position of the queue job.

jobControlFlags

Indicates the set of queue job control flags affecting the way a queue server processes a queue job. This member is used only by the DOS Requester and OS/2. Returns 0 under DOS/Windows. Under OS/2 *jobControlFlags* is defined as 0x0400 (print interrupted capture), and bits 0, 1, and 2 must be 0.

jobFileName

Indicates the name of the job file (in DOS 8.3 format) created by Queue Management System Service. The name of the file is in the form Q$XXXX.YYY.

jobFileHandle

Indicates the handle of the job file created by Queue Management

System Service. The name of the file is in the form Q$XXXX.YYY. It is normally not used by the client. The Queue Service provides the client with a file handle specific to the local file system (DOS or OS/2) for accessing the job.

servicingServerStation

Indicates the server processing the job. If the job isn't being processed, these values are undefined.

servicingServerTask

Indicates the task number of the queue server servicing the job. If the job isn't being processed, this value is undefined.

servicingServerID

Indicates the server. If the job isn't being processed, this value is undefined.

QueueJobListReply

Service: Queue Management
Defined In: nwqms.h

Structure

```
typedef struct
{
   nuint32   totalQueueJobs;
   nuint32   replyQueueJobNumbers;
   nuint32   jobNumberList[250];
} QueueJobListReply;
```

Pascal Structure

```
Defined in nwqms.inc

QueueJobListReply = Record
    totalQueueJobs : nuint32;
    replyQueueJobNumbers : nuint32;
    jobNumberList : Array[0..249] Of nuint32
  End;
```

Fields

totalQueueJobs

replyQueueJobNumbers

jobNumberList

QueueJobStruct

Holds the information found in the queue job record

Service: Queue Management

Defined In: nwqms.h

Structure

```
typedef struct
{
   nuint8    clientStation;
   nuint8    clientTask;
   nuint32   clientID;
   nuint32   targetServerID;
   nuint8    targetExecutionTime[6];
   nuint8    jobEntryTime[6];
   nuint16   jobNumber;
   nuint16   jobType;
   nuint8    jobPosition;
   nuint8    jobControlFlags;
   nuint8    jobFileName[14];
   nuint8    jobFileHandle[6];
   nuint8    servicingServerStation;
   nuint8    servicingServerTask;
   nuint32   servicingServerID;
   nuint8    jobDescription[50];
   nuint8    clientRecordArea[152];
} QueueJobStruct;
```

Pascal Structure

```
Defined in nwqms.inc

QueueJobStruct = Record
    clientStation : nuint8;
    clientTask : nuint8;
    clientID : nuint32;
    targetServerID : nuint32;
    targetExecutionTime : Array[0..5] Of nuint8;
    jobEntryTime : Array[0..5] Of nuint8;
    jobNumber : nuint16;
    jobType : nuint16;
    jobPosition : nuint8;
    jobControlFlags : nuint8;
    jobFileName : Array[0..13] Of nuint8;
    jobFileHandle : Array[0..5] Of nuint8;
    servicingServerStation : nuint8;
    servicingServerTask : nuint8;
    servicingServerID : nuint32;
    jobDescription : Array[0..49] Of nuint8;
    clientRecordArea : Array[0..151] Of nuint8
  End;
```

Fields

clientStation

Indicates the client submitting the job to the queue.

clientTask

Indicates the task number the station was performing when it placed the job in the queue.

clientID

Indicates the object ID of the person submitting the job.

targetServerID

Indicates the server ID of the queue server servicing the job. If this field is set to 0xFFFFFFFF, any queue server can service the job. If the specified queue server is not attached to the queue, Queue Management System Service removes the job from the queue.

targetExecutionTime

Indicates the earliest time the job can be serviced. The time is in this format: year, month, day, hour, minute, second (YYMMDDHHMMSS). If set to 0xFFFFFFFFFFFF, the job is serviced at the first opportunity.

jobEntryTime

Indicates the time the client submitted the job to the queue.

jobNumber

Indicates a number used to identify the job.

jobType

Indicates the number identifying the type of job entry. A queue server can request specific job types from a queue. This value cannot be -1.

jobPosition

Indicates the job’s position in the queue. The first job is assigned position 1, the next is assigned position 2, and so on. As jobs are removed from the queue, this number changes to reflect the updated position of the queue job.

jobControlFlags

Indicates flag bits indicating the status of the job. Bits 0, 1, and 2 must be 0.

jobFileName

Indicates the name of the job file (in DOS 8.3 format) created by Queue Management System Service. The name of the file is in the form Q$XXXX.YYY where Q$ is a required expression, XXXX is the last four digits of the queue identification number, and YYY is the job number.

The calling workstation can place information destined for the queue server into the job file. The file is always created, even if the client does not use it. After the file has been written, the requesting workstation should then call **NWCloseFileAndStartQueueJob2**.

jobFileHandle

Indicates the handle of the job file created by Queue Management System Service. The name of the file is in the form Q$XXXX.YYY where Q$ is a required expression, XXXX is the last four digits of the queue identification number, and YYY is the job number. It is normally not used by the client. The Queue Service provides the client with a file handle specific to the local file system (DOS or OS/2) for accessing the job.

servicingServerStation

Indicates the server processing the job. If the job isn't being processed, these values are undefined.

servicingServerTask

Indicates the task number of the queue server servicing the job. If the job isn't being processed, this value is undefined.

servicingServerID

Indicates the server. If the job isn't being processed, this value is undefined.

jobDescription

Indicates the NULL-terminated ASCII text description of the content or purpose of a job. Queue Management System Service displays this text as part of the job description when users or operators examine a queue.

clientRecordArea

Indicates supplementary information that can be exchanged between the queue client and the queue server. The format for this information is defined by your application.

ReplyJobStruct

Service: Queue Management
Defined In: nwqms.h

Structure

```
typedef struct
{
   nuint8    clientStation;
   nuint8    clientTask;
   nuint32   clientID;
   nuint32   targetServerID;
   nuint8    targetExecutionTime[6];
   nuint8    jobEntryTime[6];
   nuint16   jobNumber;
   nuint16   jobType;
   nuint8    jobPosition;
   nuint8    jobControlFlags;
   nuint8    jobFileName[14];
   nuint8    jobFileHandle[6];
   nuint8    servicingServerStation;
   nuint8    servicingServerTask;
   nuint32   servicingServerID;
} ReplyJobStruct;
```

Pascal Structure

```
Defined in nwqms.inc

  ReplyJobStruct = Record
    clientStation : nuint8;
    clientTask : nuint8;
    clientID : nuint32;
    targetServerID : nuint32;
    targetExecutionTime : Array[0..5] Of nuint8;
    jobEntryTime : Array[0..5] Of nuint8;
    jobNumber : nuint16;
    jobType : nuint16;
    jobPosition : nuint8;
    jobControlFlags : nuint8;
    jobFileName : Array[0..13] Of nuint8;
    jobFileHandle : Array[0..5] Of nuint8;
    servicingServerStation : nuint8;
    servicingServerTask : nuint8;
    servicingServerID : nuint32
  End;
```

Fields

Fields

clientStation

Indicates the client submitting the job to the queue.

clientTask

Indicates the task number the station was performing when it placed the job in the queue.

clientID

Indicates the object ID of the person submitting the job.

targetServerID

Indicates the server ID of the queue server servicing the job. If this field is set to 0xFFFFFFFF, any queue server can service the job. If the specified queue server is not attached to the queue, Queue Management System Service removes the job from the queue.

targetExecutionTime

Indicates the earliest time the job can be serviced. The time is in the format: year, month, day, hour, minute, second (YYMMDDHHMMSS). If this field is set to 0xFFFFFFFFFFFF, the job is serviced at the first opportunity.

jobEntryTime

Indicates the time the client submitted the job to the queue.

jobNumber

Indicates a number used to identify the job.

jobType

Indicates the number identifying the type of job entry. A queue server can request specific job types from a queue. This value cannot be -1.

jobPosition

Indicates the job's position in the queue. The first job is assigned position 1, the next is assigned position 2, and so on. As jobs are removed from the queue, this number changes to reflect the updated position of the queue job.

jobControlFlags

Indicates the set of queue job control flags affecting the way a queue server processes a queue job. This member is used only by the DOS Requester and OS/2. Returns 0 under DOS/Windows. Under OS/2 *jobControlFlags* is defined as 0x0400 (print interrupted capture), and bits 0, 1, and 2 must be 0.

jobFileName

Indicates the name of the job file (in DOS 8.3 format) created by Queue Management System Service. The name of the file is in the form Q$XXXX.YYY.

jobFileHandle

Indicates the handle of the job file created by Queue Management

System Service. The name of the file is in the form Q$XXXX.YYY. It is normally not used by the client. The Queue Service provides the client with a file handle specific to the local file system (DOS or OS/2) for accessing the job.

servicingServerStation

Indicates the server processing the job. If the job isn't being processed, these values are undefined.

servicingServerTask

Indicates the task number of the queue server servicing the job. If the job isn't being processed, this value is undefined.

servicingServerID

Indicates the server. If the job isn't being processed, this value is undefined.

Server Environment

Server Environment: Guides

Server Environment: General Guide

Server Environment Introduction

NetWare 2.2 and Above Server Functions

NetWare 4.x Server Functions

NetWare 2.2 Server Information Functions

Server Environment: Concepts

Server Environment: Functions

Server Environment: Structures

Parent Topic:

Management Overview

NetWare 2.2 and Above Server Functions

This is a list of the functions found in the header file server.h. This file defines NetWare Server functions that attach and log into NetWare servers and obtain server configuration data. These functions work for NetWare 2.2 and above.

Server Connection Functions

Server Console Functions

Server Configuration Functions

Parent Topic:

Server Environment: General Guide

NetWare 4.x Server Functions

This is a list of the functions found in the header file nwfse.h. This file defines NetWare Server functions that are compatible with NetWare 4.x only.

4.x Server Information Functions

4.x Server NLM Information Functions

4.x Server LAN Board Information Functions

4.x Server Protocol Stack Information Functions

4.x Server Media Manager Information Functions

4.x Server Volume Information Functions

4.x Server Network and Router Information Functions

4.x Server Get Functions

4.x Server Set Functions

4.x Server User Information Functions

Parent Topic:

Server Environment: General Guide

Server Environment: Concepts

4.x Server Get Functions

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

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

Parent Topic:

NetWare 4.x Server Functions

4.x Server Information Functions

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

Function	Header	Comment
NWGetCacheInfo	nwfse.h	Returns information about a server's file cache.
NWGetFileServerInfo	nwfse.h	Returns server operation statistics information.
NWGetNetWareFileSystemsInfo	nwfse.h	Returns counters of the times specific operations on the file system were performed.
NWGetPacketBurstInfo	nwfse.h	Returns counters and statistics about packet burst on the server.
NWGetIPXSPXInfo	nwfse.h	Returns a server's internal

		IPX and SPX statistics.
NWGetGarbageCollectionInfo	nwfse.h	Returns counters about a server's memory allocation manager.
NWGetDirCacheInfo	nwfse.h	Returns statistics about a server's directory caching.
NWGetCPUInfo	nwfse.h	Returns CPU information and descriptive strings for the CPU type, numeric coprocessor, and bus type for the indicated CPU number (for 4.x, this is 1).
NWGetVolumeSwitchInfo	nwfse.h	Returns counters containing the number of times a specified code path was taken in the server.
NWGetOSVersionInfo	nwfse.h	Returns version information about the server's operating system.

Parent Topic:

NetWare 4.x Server Functions

4.x Server LAN Board Information Functions

These functions return information about LAN boards on a 4.x server.

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

Parent Topic:

NetWare 4.x Server Functions

4.x Server Media Manager Information Functions

These functions return media manager information for a 4.x server.

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

Parent Topic:

NetWare 4.x Server Functions

4.x Server Network and Router Information Functions

These functions return routing and service advertising information for a 4.x server.

Function	Header	Comment
NWGetKnownNetworksInfo	nwfse.h	Returns a list of networks known to a server.
NWGetNetworkRouterInfo	nwfse.h	Returns information about

NWGetNetworkRouterInfo	nwfse.h	Returns information about the specified network to a server, if known.
NWGetGeneralRouterAndSAPInfo	nwfse.h	Returns flags and information concerning the status of routing and SAP on a server.
NWGetNetworkRoutersInfo	nwfse.h	Returns information about routers on the specified network.

Parent Topic:

NetWare 4.x Server Functions

4.x Server NLM Information Functions

These functions return information about NLMs on a 4.x server.

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

Parent Topic:

NetWare 4.x Server Functions

4.x Server Protocol Stack Information Functions

These functions return protocol stack information for a 4.x server.

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

Parent Topic:

NetWare 4.x Server Functions

4.x Server Set Functions

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

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

Parent Topic:

NetWare 4.x Server Functions

4.x Server User Information Functions

These functions return user information for a 4.x server.

Function	Header	Comment
NWGetActiveConnListByType	nwfse.h	Returns a bit map list of all connections of a specified type (NCP, FTAM, AppleTalk, NLM).
NWGetUserInfo	nwfse.h	Returns user information about the specified connection, including the user's name.

Parent Topic:

NetWare 4.x Server Functions

4.x Server Volume Information Functions

These functions return volume information for a 4.x server.

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

Parent Topic:

NetWare 4.x Server Functions

Server Environment Introduction

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

Check the version of NetWare running on the server

Check whether you have console operator privileges on a server

Bring down a server

Enable and disable server logins

Attach, log in, and log out of NetWare servers using the bindery

The functions fall into three groups:

Functions compatible with 2.2 and above

Functions compatible with 4.x only

Functions compatible with 2.2 only

The header nwfse.h defines NetWare 4.x only functions; nwserver.h defines the remaining functions.

For a description of structures and other data definitions that relate to this topic, see Server Environment: Structures.

Parent Topic:

Server Environment: General Guide

Related Topics:

NetWare 2.2 and Above Server Functions

NetWare 4.x Server Functions

NetWare 2.2 Server Information Functions

NetWare 2.2 Server Information Functions

These functions return statistical information for 2.2 servers.

Function	Header	Comment
NWGetDiskCacheStats	nwserver.h	Returns statistics about disk caching on a 2.2 server.
NWGetDiskChannelStats	nwserver.h	Returns statistics about a specified disk channel on a 2.2 server.
NWGetFileServerLANIOStats	nwserver.h	Returns statistics about packets being sent and received by a 2.2 server.
NWGetFileServerMiscInfo	nwserver.h	Returns miscellaneous information about a 2.2 server.

NWGetFileSystemStats	nwserver.h	Returns statistics about the file system for a 2.2 server.
NWGetFSDriveMapTable	nwserver.h	Returns the drive mapping table for a 2.2 server.
NWGetFSLANDriverConfigInfo	nwserver.h	Returns configuration information for a specified LAN driver on a 2.2 server.
NWGetPhysicalDiskStats	nwserver.h	Returns information about a physical disk on a 2.2 server.

Parent Topic:

Server Environment: General Guide

Server Configuration Functions

These functions read NetWare® server configuration data.

Function	Header	Comment
NWCheckNetWareVersion	nwserver.h	Allows verification of compatibility between applications and the version of NetWare running on a NetWare server.
NWGetFileServerDateAndTime	nwserver.h	Returns the network date and time maintained on the specified NetWare server.
NWGetFileServerDescription	nwserver.h	Returns descriptive information about the specified NetWare server, including company name, version, revision date, and copyright notice.
NWGetFileServerExtendedInfo	nwserver.h	Returns extended information about the specified NetWare server.
NWGetFileServerInformation	nwserver.h	Returns commonly referenced information about a NetWare server including version numbers, connection statistics, SFT level, and TTS level.
NWGetFileServerLoginStatus	nwserver.h	Returns whether clients can log in to the specified

		workstation.
NWCCGetConnInfo	nwserver.h	Returns the name of the NetWare server associated with the specified connection ID.
NWGetFileServerVersionInfo	nwserver.h	Returns name and version information for the specified NetWare server.
NWGetNetworkSerialNumber	nwserver.h	Returns the NetWare server's serial number and the application number.
NWIsManager	nwserver.h	Checks whether a calling station is a manager

Parent Topic:

NetWare 2.2 and Above Server Functions

Server Connection Functions

These functions perform NetWare® server attachments and logins.

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

Parent Topic:

NetWare 2.2 and Above Server Functions

Server Console Functions

These functions perform console operations.

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

Parent Topic:

NetWare 2.2 and Above Server Functions

Server Environment: Functions

GetServerConfigurationInfo

Returns the engine type and loader type of the server
Local Servers: nonblocking
Remote Servers: N/A
NetWare Server: 3.12, 4.02, 4.1
Platform: NLM
SMP Aware: No
Service: Server Environment

Syntax

```
#include <nwenvrn.h>
int GetServerConfigurationInfo (
   int  *serverType,
   int  *loaderType);
```

Parameters

serverType

(OUT) Receives the NetWare server engine type.

loaderType

(OUT) Receives the NetWare loader type.

Return Values

	0x 00	ESUCCESS always.

Remarks

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

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

	TYPE_NORMAL_SE RVER	NLM application is running on a normal NetWare server.
	TYPE_IO_ENGINE	NLM applicaton is running in the IO Engine of a NetWare SFT III™ server.
	TYPE_OS_ENGINE	NLM application is running in the MS Engine of a NetWare SFT III server.

		Engine of a NetWare SFT III server.

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

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

Example

GetServerConfigurationInfo

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

void main()
{
   int  serverType, loaderType;

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

NWAttachToFileServer

Attaches to the specified NetWare server

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWAttachToFileServer
  (pnstr8                   serverName,
   nuint16                  scopeFlag,
   NWCONN_HANDLE N_FAR *    newConnID);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

serverName

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

scopeFlag

Is reserved; must be 0.

newConnID

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 800	ALREADY_ATTACHED
0x8 801	INVALID_CONNECTION
0x8 847	NO_SERVER_ERROR
0x8 9FC	UNKNOWN_FILE_SERVER
0x8 9FF	NO_RESPONSE_FROM_SERVER

Remarks

WARNING: Before attaching to the specified server, NWAttachToFileServer tries to get the server's net address from the default server's Bindery. Under OS/2, if it cannot find a default server, the requester attempts to find any server and look for a net address in the server's Bindery.

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

NCP Calls

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

NWAttachToFileServerByConn

Attaches to a NetWare server through a service identified by a connection
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

NWCCODE N_API NWAttachToFileServerByConn
  (NWCONN_HANDLE           conn,
   pnstr8                  serverName,
   nuint16                 scopeFlag,
   NWCONN_HANDLE N_FAR *   newConnID);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

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

serverName

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

scopeFlag

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

newConnID

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 800	ALREADY_ATTACHED
0x8 801	INVALID_CONNECTION
0x8 847	NO_SERVER_ERROR
0x8 9FC	UNKNOWN_FILE_SERVER

Remarks

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

NCP Calls

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

See Also

NWAttachToFileServer

NWCheckConsolePrivileges

Determines if the logged-in user is a console operator
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: Server Environment

Syntax

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

NWCCODE N_API NWCheckConsolePrivileges (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89C6	NO_CONSOLE_PRIVILEGES

NCP Calls

0x2222 23 200   Check Console Privileges

NWCheckNetWareVersion

Checks compatibility of OS modules
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

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

minVer

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

minSubVer

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

minRev

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

minSFT

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

minTTS

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

compatibilityFlag

(OUT) Points to a flag indicating compatibility:

C Value	**Pascal Value**	**Value Name**
0x00	$00	COMPATIBLE
0x01	$01	VERSION_NUMBER_TOO_LOW
0x02	$02	SFT_LEVEL_TOO_LOW
0x04	$04	TTS_LEVEL_TOO_LOW

Return Values

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

0x0000	SUCCESSFUL

NCP Calls

0x2222 23 200 Check Console Privileges

See Also

NWGetFileServerVersionInfo

NWDetachFromFileServer (obsolete 6/96)

Breaks a workstation/NetWare server connection and relinquishes the connection number but is now obsolete. Call **NWCCGetConnInfo** followed by a call to **NWCCSysCloseConnRef** instead.

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWDetachFromFileServer
   (NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

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

0x0 000	SUCCESSFUL

Remarks

Detaching from a NetWare server is not the same as logging out from a NetWare server. Detaching relinquishes the connection number the workstation was using and breaks the connection. The shell or requester automatically removes all drives mapped to detached NetWare servers. For the workstation to send further requests to that NetWare server, it must reattach. In contrast, logging out from a NetWare server does not relinquish the connection.

To call **NWDetachFromFileServer (obsolete 6/96)**, you must have console operator rights.

NCP Calls

None

NWDisableFileServerLogin

Allows an operator to instruct the NetWare server to refuse new login 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: Server Environment

Syntax

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

NWCCODE N_API NWDisableFileServerLogin (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

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

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

To call **NWDisableFileServerLogin**, you must have console operator rights.

NCP Calls

0x2222 23 203	Disable File Server Login

NWDownFileServer

Allows a supervisor to bring down a NetWare server from a remote console

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

forceFlag

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

801	
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Down Failure

Remarks

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

To call **NWDownFileServer**, you must have console operator rights.

NCP Calls

0x2222 23 211	Down File Server

NWEnableFileServerLogin

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

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWEnableFileServerLogin (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

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

To call **NWEnableFileServerLogin**, you must have console operator rights.

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

NCP Calls

0x2222 23 204	Enable File Server Login

See Also

NWDisableFileServerLogin

NWGetActiveConnListByType

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startConnNum

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

connType

(IN) Specifies the type of the connection (NCP, AFP, NLM™, etc.).

fseActiveConnListByType

(OUT) Points to NWFSE_ACTIVE_CONN_LIST.

Return Values

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

0x00 00	SUCCESSFUL
0x89 7E	NCP_BOUNDARY_CHECK_FAILED
0x89 FD	Invalid Connection
0x89 FF	Failure or Invalid Start Number

Remarks

Console operator rights are NOT necessary to call **NWGetActiveConnListByType**.

The following connection types are defined for *connType*.

```
FSE_NCP_CONNECTION_TYPE     2
FSE_NLM_CONNECTION_TYPE     3
FSE_AFP_CONNECTION_TYPE     4
FSE_FTAM_CONNECTION_TYPE    5
FSE_ANCP_CONNECTION_TYPE      6
```

NCP Calls

0x2222 123 14   Get Active Connection List By Type

See Also

NWGetUserInfo

NWGetActiveLANBoardList

Returns information about the active LAN boards on a server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

(IN) Specifies the starting LAN board number.

fseActiveLANBoardList

(OUT) Points to NWFSE_ACTIVE_LAN_BOARD_LIST.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9FF	Failure or Invalid Start Number

Remarks

Console operator rights are NOT necessary to call **NWGetActiveLANBoardList**.

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

NCP Calls

0x2222 123 20 Active LAN Board List

See Also

NWGetLANCommonCountersInfo, **NWGetLANConfigInfo**, **NWGetLANCustomCountersInfo**, **NWGetLSLLogicalBoardStats**, **NWGetProtocolStkNumsByLANBrdNum**

NWGetActiveProtocolStacks

Returns protocol stack information in NWFSE_ACTIVE_STACKS
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

fseActiveStacks

(OUT) Points to NWFSE_ACTIVE_STACKS.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure or Invalid Start Number

Remarks

To call **NWGetActiveProtocolStacks**, you must have console operator rights.

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

NCP Calls

0x2222 123 40	Active Protocol Stacks

See Also

NWGetProtocolStackConfigInfo, **NWGetProtocolStackCustomInfo**, **NWGetProtocolStackStatsInfo**, **NWGetProtocolStkNumsByLANBrdNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetCacheInfo

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseCacheInfo

(IN) Points to NWFSE_CACHE_INFO.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

97E

Remarks

To call **NWGetCacheInfo**, you must have console operator rights.

NCP Calls

0x2222 123 01	Get Cache Information

See Also

NWGetDirCacheInfo

NWGetCPUInfo

Gets CPU and hardware configuration information about the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

CPUNum

(IN) Specifies the CPU number; 0 (zero) for NetWare 4.0.

CPUName

(OUT) Points to the ASCII string of the CPU type.

numCoprocessor

(OUT) Points to the ASCII string of whether or not a coprocessor is present.

bus

(OUT) Points to the ASCII string of the bus type.

fseCPUInfo

(OUT) Points to NWFSE_CPU_INFO.

Return Values

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

0x0 000	SUCCESSFUL
0x8 9FE	DIRECTORY_LOCKED
0x8 9FF	Failure or Invalid CPU Number

Remarks

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

Console operator rights are NOT necessary to call **NWGetCPUInfo**.

NCP Calls

0x2222 123 08   CPU Information

NWGetDirCacheInfo

Returns information about the directory cache manager

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseDirCacheInfo

(OUT) Points to NWFSE_DIR_CACHE_INFO.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetDirCacheInfo**, you must have console operator rights.

NCP Calls

0x2222 123 12 Get Directory Cache Information

NWGetDiskCacheStats

Returns statistical information about a NetWare server's caching
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

NWCCODE N_API NWGetDiskCacheStats (
   NWCONN_HANDLE            conn,
   DSK_CACHE_STATS N_FAR  *statBuffer);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetDiskCacheStats
  (conn : NWCONN_HANDLE;
   Var statBuffer : DSK_CACHE_STATS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

statBuffer

(OUT) Points to DSK_CACHE_STATS.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetDiskCacheStats**, you must have console operator rights.

NCP Calls

0x2222 23 214	Read Disk Cache Statistics

NWGetDiskChannelStats

Allows a client to get the disk channel statistics for the specified Disk Channel Number

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetDiskChannelStats (
   NWCONN_HANDLE              conn,
   nuint8                     channelNum,
   DSK_CHANNEL_STATS N_FAR  *statBuffer);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetDiskChannelStats
   (conn : NWCONN_HANDLE;
    channelNum : nuint8;
    Var statBuffer : DSK_CHANNEL_STATS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

channelNum

(IN) Specifies the channel number to get status for.

statBuffer

(OUT) Points to DSK_CHANNEL_STATS.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetDiskChannelStats**, you must have console operator rights.

NCP Calls

0x2222 23 217	Get Disk Channel Statistics

NWGetFileServerDateAndTime

Returns the network date and time maintained on the specified 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: Server Environment

Syntax

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

NWCCODE N_API NWGetFileServerDateAndTime (
   NWCONN_HANDLE   conn,
   pnuint8         pbuDateTimeBufB7);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

pbuDateTimeBufB7

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

801

Remarks

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

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

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

NOTE: The year value corresponds to the specified years:

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

NCP Calls

0x2222 20 Get File Server Date And Time

NWGetFileServerDescription

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

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

companyName

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

revision

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

revisionDate

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

copyrightNotice

(OUT) Points to the copyright notice (80 characters, optional).

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 90A	Invalid connection (NLM only)
0x8 996	SERVER_OUT_OF_MEMORY

Remarks

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

Each string is NULL-terminated.

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

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

NCP Calls

0x2222 23 201 Get File Server Description Strings

NWGetFileServerExtendedInfo

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

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

accountingVer

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

VAPVer

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

queueingVer

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

printServerVer

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

virtualConsoleVer

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

securityVer

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

internetBridgeVer

(OUT) Points to the internet bridging version number (optional).

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 996	SERVER_OUT_OF_MEMORY

Remarks

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

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

To call **NWGetFileServerExtendedInfo**, you must have console operator rights.

NCP Calls

0x2222 23 17 Get File Server Information

See Also

NWGetFileServerInformation, NWGetFileServerVersionInfo

NWGetFileServerInfo

Calls for the server's operational statistics
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseFileServerInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8	NCP_BOUNDARY_CHECK_FAILED

0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

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

Console operator rights are NOT necessary to call **NWGetFileServerInfo**.

NCP Calls

0x2222 123 02	Get File Server Information

NWGetFileServerInformation

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

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

serverName

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

majorVer

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

minVer

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

rev

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

maxConns

(OUT) Points to the maximum number of connections the server will support (optional). The connection table for 4.0 is dynamic. This number will be the maximum of what the table has grown to.

maxConnsUsed

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

connsInUse

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

numVolumes

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

SFTLevel

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

TTSLevel

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

Return Values

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

0x0000	SUCCESSFUL
0x8996	SERVER_OUT_OF_MEMORY

Remarks

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

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

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

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

NCP Calls

0x2222 23 17   Get File Server Information

See Also

NWGetFileServerInformation, **NWGetFileServerVersionInfo**

NWGetFileServerLANIOStats

Returns statistical information about packets being received and sent by a NetWare server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFileServerLANIOStats (
   NWCONN_HANDLE                    conn,
   SERVER_LAN_IO_STATS N_FAR   *statBuffer);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFileServerLANIOStats
  (conn : NWCONN_HANDLE;
   Var statBuffer : SERVER_LAN_IO_STATS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

statBuffer

(OUT) Points to SERVER_LAN_IO_STATS.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

801

Remarks

To call **NWGetFileServerLANIOStats**, you must have console operator rights.

Under 3.x and 4.x NetWare servers, ERR_NCP_NOT_SUPPORTED (0x89FB) will be returned.

NCP Calls

0x2222 23 231   Get File Server LAN I/O Statistics

NWGetFileServerLoginStatus

Returns a NetWare server's login status
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: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

loginEnabledFlag

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

801	
0x8 996	SERVER_OUT_OF_MEMORY

Remarks

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

NCP Calls

0x2222 23 205 Get File Server Login Status

NWGetFileServerMiscInfo

Returns miscellaneous information about a 2.2 NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFileServerMiscInfo
   (NWCONN_HANDLE        conn,
    NW_FS_INFO N_FAR    *fsInfo);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFileServerMiscInfo
   (conn : NWCONN_HANDLE;
    Var fsInfo : NW_FS_INFO
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fsInfo

(OUT) Points to NW_FS_INFO.

Return Values

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

0x0000	SUCCESSFUL
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetFileServerMiscInfo**, you must have console operator rights.

NCP Calls

0x2222 23 232	Get Server Misc Info

NWGetFileServerName (obsolete 6/96)

Returns the name of the specified NetWare server but is now obsolete. Call **NWCCGetConnInfo** instead.

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFileServerName
  (NWCONN_HANDLE   conn,
   pnstr8          serverName);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFileServerName
  (conn : NWCONN_HANDLE;
   serverName : pnstr8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

serverName

(OUT) Points to the NetWare server name (maximum length = 49 bytes).

Return Values

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

0x0 000	SUCCESSFUL
0x8	INVALID_CONNECTION

0x8 801	INVALID_CONNECTION

NCP Calls

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

See Also

NWCCGetConnInfo, **NWGetConnectionInformation**

NWGetFileServerVersion (obsolete 6/96)

Returns the NetWare server version multiplied by 1000 but is now obsolete. Call **NWCCGetConnInfo** instead.

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFileServerVersion
   (NWCONN_HANDLE   conn,
    pnuint16        serverVersion);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFileServerVersion
   (conn : NWCONN_HANDLE;
    serverVersion : pnuint16
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

serverVersion

(OUT) Points to the NetWare server version multiplied by 1000.

Return Values

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

0x0 000	SUCCESSFUL
0x8 80	INVALID_CONNECTION

80	
0x8 99	SERVER_OUT_OF_MEMORY

Remarks

For example, **NWGetFileServerVersion (obsolete 6/96)** returns 3110 if the NetWare version is 3.11. **NWGetFileServerVersion (obsolete 6/96)** always returns the Personal NetWare version as a nunber less than 2000.

To call **NWGetFileServerVersion (obsolete 6/96)**, you must have console operator rights.

NCP Calls

0x2222 23 17   Get File Server Information

NWGetFileServerVersionInfo

Returns information about a NetWare server's name and version levels
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: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn
(IN) Specifies the NetWare server connection handle.

versBuffer
(OUT) Points to VERSION_INFO.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

0x8996	SERVER_OUT_OF_MEMORY

Remarks

To call **NWGetFileServerVersionInfo**, you must have console operator rights.

NCP Calls

0x2222 23 17	Get File Server Information

NWGetFileSystemStats

Returns statistics about the file system for a NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFileSystemStats
   (NWCONN_HANDLE          conn,
    FILESYS_STATS N_FAR  *statBuffer);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFileSystemStats
   (conn : NWCONN_HANDLE;
    Var statBuffer : FILESYS_STATS
) : NWCCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

statBuffer
(OUT) Points to FILESYS_STATS.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetFileSystemStats**, you must have console operator rights.

NCP Calls

0x2222 23 212   Get File Systems Statistics

NWGetFSDriveMapTable

Returns NetWare server's Drive Mapping Table
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFSDriveMapTable
   (NWCONN_HANDLE           conn,
    DRV_MAP_TABLE N_FAR   *tableBuffer);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFSDriveMapTable
   (conn : NWCONN_HANDLE;
    Var tableBuffer : DRV_MAP_TABLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

tableBuffer

(OUT) Points to DRV_MAP_TABLE.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetFSDriveMapTable**, you must have console operator rights.

If the calling station does not have operator privileges, No Console Rights is returned.

NCP Calls

0x2222 23 215 Get Drive Mapping Table

NWGetFSLANDriverConfigInfo

Returns configuration information for a specified LAN driver installed on a NetWare server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetFSLANDriverConfigInfo (
   NWCONN_HANDLE         conn,
   nuint8                lanBoardNum,
   NWLAN_CONFIG N_FAR  *lanConfig);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetFSLANDriverConfigInfo
  (conn : NWCONN_HANDLE;
   lanBoardNum : nuint8;
   Var lanConfig : NWLAN_CONFIG
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

lanBoardNum

(IN) Specifies the physical LAN board number.

lanConfig

(OUT) Points to NWLAN_CONFIG.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

Remarks

To call **NWGetFSLANDriveConfigInfo**, you must have console operator rights.

NCP Calls

0x2222 23 227	Get LAN Driver Configuration Information

NWGetGarbageCollectionInfo

Returns the server's memory manager's statistical information
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseGarbageCollectionInfo

(OUT) Points to NWFSE_GARBAGE_COLLECTION_INFO returning garbage collection information.

Return Values

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

0x0 000	SUCCESSFUL
0x8	NCP_BOUNDARY_CHECK_FAILED

0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetGarbageCollectionInfo**, you must have console operator rights.

NCP Calls

0x2222 123 07	Garbage Collection Information

NWGetGeneralRouterAndSAPInfo

Returns router information received from RIP and SAP packets
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseGeneralRouterSAPInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8	INVALID_CONNECTION

0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetGeneralRouterAndSAPInfo**, you must have console operator rights.

NCP Calls

0x2222 123 50   Get General Router and SAP Information

NWGetIPXSPXInfo

Returns the server's internal IPX™ and SPX™ statistics information
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn
(IN) Specifies the NetWare server connection handle.

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

97E	

Remarks

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

Console operator rights are NOT necessary to call **NWGetIPXSPXInfo**.

NCP Calls

0x2222 123 06   IPX SPX Information

NWGetKnownNetworksInfo

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

fseKnownNetworksInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetKnownNetworksInfo**, you must have console operator rights.

NCP Calls

0x2222 123 53 Get Known Networks Information

See Also

NWGetNetworkRouterInfo, **NWGetNetworkRoutersInfo**

NWGetKnownServersInfo

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

serverType

(OUT) Receives the server type: 0x0400 = NetWare server; 0xFFFF = All Types Of Servers.

fseKnownServerInfo

(OUT) Points to NWFSE_KNOWN_SERVER_INFO containing known server information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 901	Returned from NWFSE_KNOWN_SERVER_INFO when no more items are found
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetKnownServersInfo**, you must have console operator rights.

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

NCP Calls

0x2222 123 56 Get Known Servers Information

See Also

NWGetServerSourcesInfo

NWGetLANCommonCountersInfo

Returns common statistics for a LAN board

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

boardNum

(IN) Specifies the board numbers returned by **NWGetActiveLANBoardList**.

blockNum

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

fseLANCommonCountersInfo

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

Return Values

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

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x897E	NCP_BOUNDARY_CHECK_FAILED
0x89C6	NO_CONSOLE_PRIVILEGES
0x89FF	Failure or Invalid Board or Block Number

Remarks

To call **NWGetLANCommonCountersInfo**, you must have console operator rights.

NCP Calls

0x2222 123 22 LAN Common Counters Information

See Also

NWGetActiveLANBoardList, **NWGetLANConfigInfo**, **NWGetLANCustomCountersInfo**, **NWGetLSLLogicalBoardStats**, **NWGetProtocolStkNumsByLANBrdNum**

NWGetLANConfigInfo

Returns configuration information for the LAN board identified by *boardNum*

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

boardNum

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

fseLANConfigInfo

(OUT) Points to NWFSE_LAN_CONFIG_INFO returning LAN configuration information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 979	ERR_NO_ITEMS_FOUND
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure or Invalid Board Number

Remarks

To call **NWGetLANConfigInfo**, you must have console operator rights.

NCP Calls

0x2222 123 21 LAN Configuration Information

See Also

NWGetActiveLANBoardList, **NWGetLANCommonCountersInfo**, **NWGetLANCustomCountersInfo**, **NWGetLSLLogicalBoardStats**, **NWGetProtocolStkNumsByLANBrdNum**

NWGetLANCustomCountersInfo

Returns custom statistics for a LAN board

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

boardNum

(IN) Specifies the board number returned by **NWGetActiveLANBoardList**.

startingNum

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

fseLANCustomInfo

(OUT) Points to NWFSE_LAN_CUSTOM_INFO returning information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure or Invalid Start or Board Number

Remarks

To call **NWGetLANCustomCountersInfo**, you must have console operator rights.

NCP Calls

0x2222 123 23   LAN Custom Counters Information

See Also

NWGetActiveLANBoardList, **NWGetLANConfigInfo**, **NWGetLANCommonCountersInfo**, **NWGetLSLLogicalBoardStats**, **NWGetProtocolStkNumsByLANBrdNum**

NWGetLoadedMediaNumList

Returns a list of loaded media numbers
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseLoadedMediaNumList

(OUT) Points to NWFSE_LOADED_MEDIA_NUM_LIST returning information.

Return Values

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

0x0 000	SUCCESSFUL
0x8	INVALID_CONNECTION

0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetLoadedMediaNumList**, you must have console operator rights.

NCP Calls

0x2222 123 47 Get Loaded Media Num List

See Also

NWGetMediaMgrObjChildrenList, **NWGetMediaMgrObjInfo**, **NWGetMediaMgrObjList**, **NWGetMediaNameByMediaNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetLSLInfo

Returns LSL information

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseLSLInfo

(OUT) Points to NWFSE_LSL_INFO returning LSL™ information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetLSLInfo**, you must have console operator rights.

NCP Calls

0x2222 123 25   LSL Information

See Also

NWGetLSLLogicalBoardStats

NWGetLSLLogicalBoardStats

Returns LSL logical board statistics
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

LANBoardNum

(IN) Specifies a board number returned by **NWGetActiveLANBoardList**.

fseLSLLogicalBoardStats

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

Return Values

These are common return values; see Return Values for more

information.

0x00 000	SUCCESSFUL
0x88 01	INVALID_CONNECTION
0x89 7E	NCP_BOUNDARY_CHECK_FAILED
0x89 C6	NO_CONSOLE_PRIVILEGES
0x89 FB	ERR_NCP_NOT_SUPPORTED
0x89 FF	Failure or Invalid LAN Board Number

Remarks

To call **NWGetLSLLogicalBoardStats**, you must have console operator rights.

NCP Calls

0x2222 123 26   LSL Logical Board Statistics

See Also

NWGetLSLInfo

NWGetMediaMgrObjChildrenList

Returns a list of children belonging to a given media manager parent object
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

objType

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

parentObjNum

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

fseMediaMgrObjList

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

Return Values

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

0x00 000	SUCCESSFUL
0x89 79	Invalid Start Number, Object Type, or Parent Object Number
0x89 7E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetMediaMgrObjChildrenList**, you must have console operator rights.

NCP Calls

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

See Also

NWGetLoadedMediaNumList, **NWGetMediaMgrObjInfo**, **NWGetMediaMgrObjList**, **NWGetMediaNameByMediaNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetMediaMgrObjInfo

Returns information about media manager objects (storage devices)

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objNum

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

fseMediaMgrObjInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetMediaMgrObjInfo**, you must have console operator rights.

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

NCP Calls

0x2222 123 30	Get Media Manager Object Information

See Also

NWGetLoadedMediaNumList, **NWGetMediaMgrObjChildrenList**, **NWGetMediaMgrObjList**, **NWGetMediaNameByMediaNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetMediaMgrObjList

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

(IN) Specifies the value returned in the *nextStartObjNum* field of the *fseMediaMgrObjList* parameter (set to zero initially).

objType

(IN) Specifies the object type.

fseMediaMgrObjList

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 979	Invalid Start Number or Object Type
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetMediaMgrObjList**, you must have console operator rights.

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

Object types are the following:

C Valu e	**Pascal Value**	**Value Name**
0	0	FSE_ADAPTER_OBJECT
1	1	FSE_CHANGER_OBJECT
2	2	FSE_DEVICE_OBJECT
4	4	FSE_MEDIA_OBJECT
5	5	FSE_PARTITION_OBJECT
6	6	FSE_SLOT_OBJECT
7	7	FSE_HOTFIX_OBJECT
8	8	FSE_MIRROR_OBJECT
9	9	FSE_PARITY_OBJECT
10	10	FSE_VOLUME_SEG_OBJECT
11	11	FSE_VOLUME_OBJECT
12	12	FSE_CLONE_OBJECT
14	14	FSE_MAGAZINE_OBJECT
15	15	FSE_VIRTUAL_DEVICE_OBJECT

0xFF FF	$FFFF	FSE_UNKNOWN_OBJECT

NCP Calls

0x2222 123 31 Get Media Manager Objects List

See Also

NWGetLoadedMediaNumList, **NWGetMediaMgrObjInfo**, **NWGetMediaMgrObjChildrenList**, **NWGetMediaNameByMediaNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetMediaNameByMediaNum

Returns the identifying name or label for the given media object

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

mediaNum

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

mediaName

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

fseMediaNameList

(OUT) Points to NWFSE_MEDIA_NAME_LIST returning information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetMediaNameByMediaNum**, you must have console operator rights.

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

NCP Calls

0x2222 123 46	Get Media Name By Media Number

See Also

NWGetLoadedMediaNumList, **NWGetMediaMgrObjInfo**, **NWGetMediaMgrObjChildrenList**, **NWGetMediaMgrObjList**, **NWGetProtocolStkNumsByMediaNum**

NWGetNetWareFileSystemsInfo

Returns information about a loaded file system
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseFilesystemInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8	NCP_BOUNDARY_CHECK_FAILED

0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetNetWareFileSystemsInfo**, you must have console operator rights.

NCP Calls

0x2222 123 03 NetWare File Systems Information

NWGetNetworkRouterInfo

Returns information about network routing (other networks known to NetWare server)

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

networkNum

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

fseNetworkRouterInfo

(OUT) Points to NWFSE_NETWORK_ROUTER_INFO returning network router information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION

Remarks

To call **NWGetNetworkRouterInfo**, you must have console operator rights.

NCP Calls

0x2222 123 51 Get Network Router Information

See Also

NWGetKnownNetworksInfo, **NWGetNetworkRoutersInfo**

NWGetNetworkRoutersInfo

Returns information about the routers on a network

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

networkNum

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

startNum

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

fseNetworkRoutersInfo

(OUT) Points to NWFSE_NETWORK_ROUTERS_INFO returning network routers information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	Invalid Network Number or Start Number
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetNetworkRoutersInfo**, you must have console operator rights.

NCP Calls

0x2222 123 52 Get Network Routers Information

See Also

NWGetKnownNetworksInfo, **NWGetNetworkRouterInfo**

NWGetNetworkSerialNumber

Returns the NetWare server's serial number and the application number
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: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn
(IN) Specifies the NetWare server connection handle.

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

appNum
(OUT) Points to the application number.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 996	SERVER_OUT_OF_MEMORY

Remarks

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

NCP Calls

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

NWGetNLMInfo

Returns information about a specific loaded NLM

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NLMNum

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

fileName

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

NLMName

(OUT) Points to the NLM name.

copyright

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

fseNLMInfo

(OUT) Points to NWFSE_NLM_INFO.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure or Invalid NLM Number

Remarks

To call **NWGetNLMInfo**, you must have console operator rights.

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

NCP Calls

0x2222 123 11 NLM Information

See Also

NWGetNLMLoadedList, **NWGetNLMsResourceTagList**

NWGetNLMLoadedList

Returns a list of loaded NLM sequence numbers

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

fseNLMLoadedList

(OUT) Points to NWFSE_NLM_LOADED_LIST.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure

Remarks

To call **NWGetNLMLoadedList**, you must have console operator rights.

NWGetNLMLoadedList will only return information about 128 NLMs at a time. If you have more than 128 NLMs loaded, **NWGetNLMLoadedList** needs to be called iteratively. Set *startNum* to zero the first time **NWGetNLMLoadedList** is called. Set *startNum* to the number returned by the *numberNLMsLoaded* field of the NWFSE_NLM_LOADED_LIST structure each time **NWGetNLMLoadedList** is called subsequently.

NCP Calls

0x2222 123 10 Get NLM Loaded List

See Also

NWGetNLMInfo, **NWGetNLMsResourceTagList**

NWGetNLMsResourceTagList

Returns information about resources used by NLMs on a server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

NLMNum

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

startNum

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

fseNLMsResourceTagList

(OUT) Points to NWFSE_NLMS_RESOURCE_TAG_LIST.

Return Values

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

0x0 000	SUCCESSFUL

Remarks

To call **NWGetNLMsResourceTagList**, you must have console operator rights.

NCP Calls

0x2222 123 15 Get NLM's Resource Tag List

See Also

NWGetNLMInfo, **NWGetNLMLoadedList**

NWGetOSVersionInfo

Gets NetWare OS version information
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fseOSVersionInfo

(OUT) Points to NWFSE_OS_VERSION_INFO getting operating system version information.

Return Values

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

0x0 000	SUCCESSFUL

Remarks

Console operator rights are NOT necessary to call **NWGetOSVersionInfo**.

NCP Calls

0x2222 123 13   Get Operating System Version Information

NWGetPacketBurstInfo

Returns the server's packet burst operational counters and statistics
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

fsePacketBurstInfo

(OUT) Points to NWFSE_PACKET_BURST_INFO getting packet burst information.

Return Values

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

0x0 000	SUCCESSFUL
0x8	NCP_BOUNDARY_CHECK_FAILED

0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetPacketBurstInfo**, you must have console operator rights.

NCP Calls

0x2222 123 05   Packet Burst Information

NWGetPhysicalDiskStats

Returns statistics about a specified disk

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetPhysicalDiskStats (
   NWCONN_HANDLE            conn,
   nuint8                   physicalDiskNum,
   PHYS_DSK_STATS N_FAR   *tatBuffer);
```

Pascal Syntax

```
#include <nwserver.inc>

Function NWGetPhysicalDiskStats
   (conn : NWCONN_HANDLE;
    physicalDiskNum : nuint8;
    Var statBuffer : PHYS_DSK_STATS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

physicalDiskNum

(IN) Specifies the physical disk number.

statBuffer

(OUT) Points to PHYS_DSK_STATS.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 996	SERVER_OUT_OF_MEMORY
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWGetPhysicalDiskStats**, you must have console operator rights.

NCP Calls

0x2222 23 216	Read Physical Disk Statistics

NWGetProtocolStackConfigInfo

Returns configuration information about the protocols on a server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

stackNum

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

stackFullName

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

fseProtocolStkConfigInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9FF	Failure

Remarks

To call **NWGetProtocolStackConfigInfo**, you must have console operator rights.

NCP Calls

0x2222 123 41 Get Protocol Stack Configuration Information

See Also

NWGetActiveProtocolStacks, **NWGetProtocolStackCustomInfo**, **NWGetProtocolStackStatsInfo**, **NWGetProtocolStkNumsByLANBrdNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetProtocolStackCustomInfo

Returns custom information about a protocol stack on a server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

stackNum

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

customStartNum

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

fseProtocolStackCustomInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9FF	Failure

Remarks

To call **NWGetProtocolStackCustomInfo**, you must have console operator rights.

NCP Calls

0x2222 123 43 Get Protocol Stack Custom Information

See Also

NWGetActiveProtocolStacks, **NWGetProtocolStackConfigInfo**, **NWGetProtocolStackStatsInfo**, **NWGetProtocolStkNumsByLANBrdNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetProtocolStackStatsInfo

Returns the protocol statistics indicated by*stackNum* for a server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

stackNum

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

fseProtocolStkStatsInfo

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

Return Values

These are common return values; see Return Values for more

information.

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure

Remarks

To call **NWGetProtocolStackStatsInfo**, you must have console operator rights.

NCP Calls

0x2222 123 42	Get Protocol Stack Statistics Information

See Also

NWGetActiveProtocolStacks, **NWGetProtocolStackConfigInfo**, **NWGetProtocolStackCustomInfo**, **NWGetProtocolStkNumsByLANBrdNum**, **NWGetProtocolStkNumsByMediaNum**

NWGetProtocolStkNumsByLANBrdNum

Returns a list of protocol stack ID numbers for a given LAN board
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

LANBoardNum

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

fseProtocolStkIDNums

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FF	Failure

Remarks

To call **NWGetProtocolStkNumsByLANBrdNum**, you must have console operator rights.

NCP Calls

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

See Also

NWGetActiveLANBoardList, **NWGetActiveProtocolStacks**, **NWGetProtocolStackConfigInfo**, **NWGetProtocolStackCustomInfo**, **NWGetProtocolStackStatsInfo**, **NWGetProtocolStkNumsByMediaNum**

NWGetProtocolStkNumsByMediaNum

Returns a list of protocol stack ID numbers for a given media
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

mediaNum

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

fseProtocolStkIDNums

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

Return Values

These are common return values; see Return Values for more

information.

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES

An Invalid Media Number can abend the server

Remarks

To call **NWGetProtocolStkNumsByMediaNum**, you must have console operator rights.

NCP Calls

0x2222 123 44	Get Protocol Stack Number By Media Number

See Also

NWGetActiveProtocolStacks, **NWGetLoadedMediaNumList**, **NWGetProtocolStackConfigInfo**, **NWGetProtocolStackCustomInfo**, **NWGetProtocolStackStatsInfo**, **NWGetProtocolStkNumsByLANBrdNum**

NWGetServerInfo

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetServerInfo (
   NWCONN_HANDLE              conn,
   nuint32                    serverType,
   pnstr8                     serverName,
   NWFSE_SERVER_INFO N_FAR  *fseServerInfo);
```

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

serverType

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

serverName

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

fseServerInfo

(OUT) Points to NWFSE_SERVER_INFO getting server information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	Invalid Server Name or Server Type
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9C6	NO_CONSOLE_PRIVILEGES
0x8 9FB	INVALID_PARAMETERS

Remarks

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

NCP Calls

0x2222 123 54 Get Server Information

See Also

NWGetServerSourcesInfo

NWGetServerSetCategories

Returns SET console command configuration parameter categories for the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

fseServerSetCategories

(OUT) Points to NWFSE_SERVER_SET_CATEGORIES getting server set categories.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9F5	Invalid Star Number

Remarks

To call **NWGetServerSetCategories**, you must have console operator rights.

NCP Calls

0x2222 123 61 Get Server Set Categories

See Also

NWGetServerSetCommandsInfo, **NWSMSetDynamicCmdIntValue**, **NWSMSetDynamicCmdStrValue**

NWGetServerSetCommandsInfo

Returns SET console command configuration parameter commands for the server

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

fseServerSetCmdsInfo

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9FF	Failure of Invalid Start Number

Remarks

To call **NWGetServerSetCommandsInfo**, you must have console operator rights.

NCP Calls

0x2222 123 60 Get Server Set Commands Information

See Also

NWGetServerSetCategories, **NWSMSetDynamicCmdIntValue**, **NWSMSetDynamicCmdStrValue**

NWGetServerSourcesInfo

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWGetServerSourcesInfo (
   NWCONN_HANDLE                    conn,
   nuint32                          startNum,
   nuint32                          serverType,
   pnstr8                           serverName,
   NWFSE_SERVER_SRC_INFO N_FAR   *fseServerSrcInfo);
```

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

serverType

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

serverName

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

fseServerSetCmdsInfo

(OUT) Points to NWFSE_SERVER_SRC_INFO getting server sources information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	Invalid Server Name or Server Type
0x8 97E	NCP_BOUNDARY_CHECK_FAILED

Remarks

To call **NWGetServerSourcesInfo**, you must have console operator rights.

NCP Calls

0x2222 123 55 Get Server Sources Information

See Also

NWGetServerInfo

NWGetUserInfo

Gets NetWare user information about a given logged-in server connection
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

connNum

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

userName

(OUT) Points to the user name; minimum buffer size is 48 characters.

fseServerSetCmdsInfo

(OUT) Points to NWFSE_USER_INFO getting user information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9FF	Failure or Invalid Connection Number

Remarks

To call **NWGetUserInfo**, you must have console operator rights.

NCP Calls

0x2222 123 04   User Information

See Also

NWCCGetAllConnInfo, **NWGetActiveConnListByType**, **NWGetConnectionInformation**

NWGetVolumeInfoByLevel

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volNum

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

infoLevel

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

fseVolumeInfo

(OUT) Points to NWFSE_VOLUME_INFO_BY_LEVEL containing the information.

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 998	VOLUME_DOES_NOT_EXIST
0x8 9FF	Failure

Remarks

Console operator rights are NOT necessary to call **NWGetVolumeInfoByLevel**.

NCP Calls

0x2222 123 34   Get Volume Information By Level

See Also

NWGetVolumeInfoWithHandle, **NWGetVolumeInfoWithNumber**

NWGetVolumeSegmentList

Returns a list of up to 32 volume segments for a given volume
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

volumeNum

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

fseVolumeSegmentList

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

Return Values

These are common return values; see Return Values for more

information.

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 998	VOLUME_DOES_NOT_EXIST

Remarks

Console operator rights are NOT necessary to call **NWGetVolumeSegmentList**.

NCP Calls

0x2222 123 33   Get Volume Segment List

NWGetVolumeSwitchInfo

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

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 4.x

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

Service: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwfse.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

startNum

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

fseVolumeSwitchInfo

(OUT) Points to NWFSE_VOLUME_SWITCH_INFO getting volume switch information.

Return Values

These are common return values; see Return Values for more

information.

0x0 000	SUCCESSFUL
0x8 97E	NCP_BOUNDARY_CHECK_FAILED
0x8 9FF	Failure or Invalid Start Item Number

Remarks

To call **NWGetVolumeSwitchInfo**, you must have console operator rights.

NCP Calls

0x2222 123 09   Volume Switch Information

NWIsManager

Checks whether a calling station is a manager

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWIsManager (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

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

0x0000	Calling station is a manager.
0x89FF	Calling station is not a manager.

Remarks

To call **NWIsManager**, you must have console operator rights.

To call **NWIsManager**, you must have console operator rights.

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

NCP Calls

0x2222 23 73   Is Calling Station A Manager

NWLoginToFileServer

Attempts to log an object in to the specified NetWare server

Local Servers: blocking

Remote Servers: blocking

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWLoginToFileServer (
   NWCONN_HANDLE   conn,
   pnstr8          objName,
   nuint16         objType,
   pnstr8          password);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

objName

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

objType

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

password

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

Return Values

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

0x00 00	SUCCESSFUL
0x88 00	ALREADY_ATTACHED
0x88 01	INVALID_CONNECTION
0x89 96	SERVER_OUT_OF_MEMORY
0x89 C1	LOGIN_DENIED_NO_ACCOUNT_BALANCE
0x89 C2	LOGIN_DENIED_NO_CREDIT
0x89 C5	INTRUDER_DETECTION_LOCK
0x89 D9	ERR_MAX_SERVERS
0x89 DA	UNAUTHORIZED_LOGIN_TIME
0x89 DB	UNAUTHORIZED_LOGIN_STATION
0x89 DC	ACCOUNT_DISABLED
0x89 DE	PASSWORD_HAS_EXPIRED_NO_GRACE
0x89 DF	PASSWORD_EXPIRED
0x89 FB	INVALID_PARAMETERS
0x89 FE	BINDERY_LOCKED
0x89 FF	NO_SUCH_OBJECT_OR_BAD_PASSWORD
0xFF	Failure

Remarks

To login to a 3.x NetWare server, both the *conn* and *password* parameters must be specified by upper case letters.

Some utilities require an upper case password to be passed to the *password* parameter.

If a Directory Services attachment exists, **NWLoginToFileServer** fails with 0x8800.

Before calling **NWLoginToFileServer**, call the **NWCCGetConnInfo** and **NWGetConnectionInformation** functions and check CONNECTION_NDS. If it is set, **NWLoginToFileServer** cannot be used to log in. **NWCCSysCloseConnRef** and **NWCCGetConnInfo** will also be prevented from being called. Call Directory Services functions to continue the operation.

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

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

Valid Bindery object types for OT_ constants follow:

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

Extended bindery object types follow:

```
OT_TIME_SYNCHRONIZATION_SERVER   0x2D00
OT_ARCHIVE_SERVER_DYNAMIC_SAP    0x2E00
```

```
  OT_ADVERTISING_PRINT_SERVER      0x4700
  OT_PRINT_QUEUE_USER              0x5300
```

NCP Calls

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

NWLogoutFromFileServer

Attempts to log the workstation out of the specified NetWare server

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

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

Service: Server Environment

Syntax

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

NWCCODE N_API NWLogoutFromFileServer (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 8FB	INVALID_PARAMETERS

Remarks

To call **NWLogoutFromFileServer**, you must have console operator rights.

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

For OS/2, after calling NetWare IFS to delete any drive mappings dependent on the connection, the requester performs the logout function.

NCP Calls

None

NWSetFileServerDateAndTime

Sets the date and time 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: Server Environment

Syntax

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

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

Pascal Syntax

```
#include <nwserver.inc>

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

Parameters

conn

(IN) Specifies the NetWare server connection handle.

year

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

month

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

day

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

hour

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

minute

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

second

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

Return Values

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

0x0 000	SUCCESSFUL
0x8 801	INVALID_CONNECTION
0x8 9C6	NO_CONSOLE_PRIVILEGES

Remarks

To call **NWSetFileServerDateAndTime**, you must have console operator rights.

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

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

NCP Calls

0x2222 23 202	Set File Server Date And Time

Server Environment: Structures

CACHE_COUNTERS

Returns cache information

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

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

Pascal Structure

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

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

Fields

readExistingBlockCount

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

readExistingWriteWaitCount

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

readExistingPartialReadCount

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

readExistingReadErrorCount

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

writeBlockCount

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

writeEntireBlockCount

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

getDiskCount

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

getDiskNeedToAllocCount

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

getDiskSomeoneBeatMeCount

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

getDiskPartialReadCount

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

getDiskReadErrorCount

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

getAsyncDiskCount

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

getAsyncDiskNeedToAlloc

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

getAsyncDiskSomeoneBeatMe

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

errorDoingAsyncReadCount

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

getDiskNoReadCount

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

getDiskNoReadAllocCount

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

getDiskNoReadSomeoneBeatMeCount

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

diskWriteCount

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

diskWriteAllocCount

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

diskWriteSomeoneBeatMeCount

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

writeErrorCount

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

waitOnSemaphoreCount

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

allocBlockWaitForSomeoneCount

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

allocBlockCount

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

allocBlockWaitCount

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

CACHE_INFO

Returns information about a cache

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

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

Pascal Structure

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

Fields

maxByteCount

Specifies the length in bytes of a cache block.

minNumOfCacheBuffers

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

minCacheReportThreshold

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

allocWaitingCount

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

numDirtyBlocks

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

cacheDirtyWaitTime

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

cacheMaxConcurrentWrites

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

maxDirtyTime

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

numOfDirCacheBuffers

Specifies the number of directory cache buffers on the server.

cacheByteToBlockShiftFactor

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

Remarks

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

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

The block size (in bytes) is calculated using:

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

where n is the shift factor.

CACHE_MEM_COUNTERS

Returns cache memory information

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

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

Pascal Structure

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

Fields

originalNumOfCacheBuffers

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

currentNumOfCacheBuffers

Specifies the number of cache buffers currently on the server.

cacheDirtyBlockThreshold

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

waitNodeCount

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

waitNodeAllocFailureCount

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

moveCacheNodeCount

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

moveCacheNodeFromAvailCount

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

accelerateCacheNodeWriteCount

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

removeCacheNodeCount

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

removeCacheNodeFromAvailCount

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

CACHE_TREND_COUNTERS

Returns cache trend information

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

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

Pascal Structure

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

Fields

numCacheChecks

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

numCacheHits

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

numDirtyCacheChecks

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

numDirtyCacheHits

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

cacheUsedWhileChecking

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

waitForDirtyBlocksDecreaseCount

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

allocBlockFromAvailCount

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

allocBlockFromLRUCount

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

allocBlockAlreadyWaiting

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

LRUSittingTime

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

CPU_INFO

Returns CPU information

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   nuint32   pageTableOwnerFlag;
   nuint32   CPUTypeFlag;
   nuint32   coProcessorFlag;
   nuint32   busTypeFlag;
   nuint32   IOEngineFlag;
   nuint32   FSEngineFlag;
   nuint32   nonDedicatedFlag;
   nuint32   cpuString;
   nuint32   numericCoprocessorPresent/NotPresentString;
   nuint32   busString;
} CPU_INFO;
```

Pascal Structure

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

Fields

pageTableOwnerFlag

Specifies which domain is the current domain.

CPUTypeFlag

Specifies the CPU type:

0=80386
1=80486
2=Pentium

coProcessorFlag

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

busTypeFlag

Specifies the bus type:

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

IOEngineFlag

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

FSEngineFlag

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

nonDedicatedFlag

Specifies whether the CPU is dedicated.

cpuString

Specifies the CPU type:

Default=CPU Type Unknown
0=80386
1=80486
2-Pentium

numericCoprocessorPresent/NotPresentString

Specifies whether a numerical coprocessor is present:

true=numerical coprocessor is present
false=numerical coprocessor is not present

busString

Specifies the type of bus:

PCI bus
PCMCIA bus
Micro channel bus
EISA bus
ISA bus
Unknown bus type

DIR_CACHE_INFO

Returns information for a directory cache

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

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

Pascal Structure

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

Fields

minTimeSinceFileDelete

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

absMinTimeSinceFileDelete

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

minNumOfDirCacheBuffers

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

maxNumOfDirCacheBuffers

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

numOfDirCacheBuffers

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

dCMinNonReferencedTime

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

dCWaitTimeBeforeNewBuffer

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

dCMaxConcurrentWrites

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

dCDirtyWaitTime

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

dCDoubleReadFlag

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

mapHashNodeCount

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

spaceRestrictionNodeCount

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

trusteeListNodeCount

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

percentOfVolumeUsedByDirs

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

Remarks

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

DRV_MAP_TABLE

Returns drive map table data

Service: Server Environment

Defined In: nwserver.h

Structure

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

Pascal Structure

```
Defined in nwserver.inc

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

Fields

systemElapsedTime

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

SFTSupportLevel

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

logicalDriveCount

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

physicalDriveCount

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

diskChannelTable

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

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

pendingIOCommands

Specifies the number of outstanding disk controller commands.

driveMappingTable

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

driveMirrorTable

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

deadMirrorTable

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

reMirrorDriveNumber

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

reserved

Is currently not used.

reMirrorCurrentOffset

Specifies the block number that is currently being remirrored.

SFTErrorTable

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

DSK_CACHE_STATS

Returns disk caching statistics

Service: Server Environment

Defined In: nwserver.h

Structure

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

Pascal Structure

```
Defined in nwserver.inc

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

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

Fields

systemElapsedTime

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

cacheBufferCount

Specifies the number of cache buffers in the server.

cacheBufferSize

Specifies the number of bytes in a cache buffer.

dirtyCacheBuffers

Specifies the number of cache buffers in use.

cacheReadRequests

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

cacheWriteRequests

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

cacheHits

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

cacheMisses

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

physicalReadReqeusts

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

physicalWriteRequests

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

physicalReadRequests

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

physicalWriteErrors

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

cacheGetRequests

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

cacheFullWriteRequests

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

cachePartialWriteRequests

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

backgroundDirtyWrites

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

backgroundAgedWrites

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

totalCacheWrites

Specifies the total number of cache buffers written to disk.

cacheAllocations

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

thrashingCount

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

LRUBlockWasDirtyCount

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

readBeyondWriteCount

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

fragmentedWriteCount

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

cacheHitOnUnavailCount

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

cacheBlockScrappedCount

Specifies the number of times a cache block was scrapped.

DSK_CHANNEL_STATS

Returns the disk channel statistics

Service: Server Environment

Defined In: nwserver.h

Structure

```
typedef struct
{
   nuint32   systemElapsedTime;
   nuint16   channelState;
   nuint16   channelSyncState;
   nuint8    driverType;
   nuint8    driverMajorVersion;
   nuint8    driverMinorVersion;
   nuint8    driverDescription[65];
   nuint16   IOAddr1;
   nuint16   IOAddr1Size;
   nuint16   IOAddr2;
   nuint16   IOAddr2Size;
   nuint8    sharedMem1Seg[3];
   nuint16   sharedMem1Ofs;
   nuint8    sharedMem2Seg[3];
   nuint16   sharedMem2Ofs;
   nuint8    interrupt1Used;
   nuint8    interrupt1;
   nuint8    interrupt2Used;
   nuint8    interrupt2;
   nuint8    DMAChannel1Used;
   nuint8    DMAChannel1;
   nuint8    DMAChannel2Used;
   nuint8    DMAChannel2;
   nuint16   reserved2;
   nuint8    configDescription[80];
} DSK_CHANNEL_STATS;
```

Pascal Structure

```
Defined in nwserver.inc

 DSK_CHANNEL_STATS = Record
    systemElapsedTime : nuint32;
    channelState : nuint16;
    channelSyncState : nuint16;
    driverType : nuint8;
    driverMajorVersion : nuint8;
    driverMinorVersion : nuint8;
    driverDescription : Array[0..64] Of nuint8;
```

```
      IOAddr1 : nuint16;
      IOAddr1Size : nuint16;
      IOAddr2 : nuint16;
      IOAddr2Size : nuint16;
      sharedMem1Seg : Array[0..2] Of nuint8;
      sharedMem1Ofs : nuint16;
      sharedMem2Seg : Array[0..2] Of nuint8;
      sharedMem2Ofs : nuint16;
      interrupt1Used : nuint8;
      interrupt1 : nuint8;
      interrupt2Used : nuint8;
      interrupt2 : nuint8;
      DMAChannel1Used : nuint8;
      DMAChannel1 : nuint8;
      DMAChannel2Used : nuint8;
      DMAChannel2 : nuint8;
      reserved2 : nuint16;
      configDescription : Array[0..79] Of nuint8
   End;
```

Fields

systemElapsedTime

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

channelState

Specifies the state of the disk channel:

```
0x00  Channel is running
0x01  Channel is stopping
0x02  Channel is stopped
0x03  Channel is not functional
```

channelSyncState

Specifies the control state of the disk channel can have the values below:

```
0x00  Channel is not being used
0x02  NetWare is using the channel; noone else wants it
0x04  NetWare is using the channel; someone else wants it
0x06  Someone else is using the channel; NetWare does not need it
0x08  Someone else is using the channel; NetWare needs it
0x0A  Someone else has released the channel; NetWare should use it.
```

driverType

Specifies which type of disk driver software is installed in the disk channel.

driverMajorVersion

Specifies the major version of the disk driver software installed on the disk channel.

driverMinorVersion

Specifies the minor version of the disk driver software installed on the disk channel.

driverDescription

Specifies the NULL-terminated string describing the disk driver software.

IOAddr1

Specifies the address the disk driver uses to control the disk channel.

IOAddr1Size

IOAddr2

Specifies the address the disk driver uses to control the disk channel.

IOAddr2Size

sharedMem1Seg

Specifies the shared memory address.

sharedMem1Ofs

Specifies the shared memory address offset.

sharedMem2Seg

Specifies the shared memory address.

sharedMem2Ofs

Specifies the shared memory address offset.

interrupt1Used

Specifies the interrupt number the disk driver uses to communicate with the disk channel.

interrupt1

interrupt2Used

Specifies the interrupt number the disk driver uses to communicate with the disk channel.

interrupt2

DMAChannel1Used

Specifies the DMA controller used by the disk driver to control the disk channel.

DMAChannel1

DMAChannel2Used

Specifies the DMA controller used by the disk driver to control the disk channel.

DMAChannel2

reserved2

configDescription

Specifies the NULL-terminated string containing the channel's current IO driver configuration.

FILE_SERVER_COUNTERS

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

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

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

Pascal Structure

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

Fields

tooManyHops

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

unknownNetwork

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

noSpaceForService

Is reserved (pass 0).

noReceiveBuffers

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

buffers existed to receive it.

notMyNetwork

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

netBIOSProgatedCount

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

totalPacketsServiced

Specifies the total packets received by the server.

totalPacketsRouted

Specifies the number of all packets forwarded by the server.

FILESYS_STATS

Returns file system statistics

Service: Server Environment

Defined In: nwserver.h

Structure

```
typedef struct
{
   nuint32   systemElapsedTime;
   nuint16   maxOpenFiles;
   nuint16   maxFilesOpened;
   nuint16   currOpenFiles;
   nuint32   totalFilesOpened;
   nuint32   totalReadRequests;
   nuint32   totalWriteRequests;
   nuint16   currChangedFATSectors;
   nuint32   totalChangedFATSectors;
   nuint16   FATWriteErrors;
   nuint16   fatalFATWriteErrors;
   nuint16   FATScanErrors;
   nuint16   maxIndexFilesOpened;
   nuint16   currOpenIndexedFiles;
   nuint16   attachedIndexFiles;
   nuint16   availableIndexFiles;
} FILESYS_STATS;
```

Pascal Structure

```
Defined in nwserver.inc

  FILESYS_STATS = Record
    systemElapsedTime : nuint32;
    maxOpenFiles : nuint16;
    maxFilesOpened : nuint16;
    currOpenFiles : nuint16;
    totalFilesOpened : nuint32;
    totalReadRequests : nuint32;
    totalWriteRequests : nuint32;
    currChangedFATSectors : nuint16;
    totalChangedFATSectors : nuint32;
    FATWriteErrors : nuint16;
    fatalFATWriteErrors : nuint16;
    FATScanErrors : nuint16;
    maxIndexFilesOpened : nuint16;
    currOpenIndexedFiles : nuint16;
    attachedIndexFiles : nuint16;
    availableIndexFiles : nuint16
```

```
   End;
```

Fields

systemElapsedTime

maxOpenFiles

maxFilesOpened

currOpenFiles

totalFilesOpened

totalReadRequests

totalWriteRequests

currChangedFATSectors

totalChangedFATSectors

FATWriteErrors

fatalFATWriteErrors

FATScanErrors

maxIndexFilesOpened

currOpenIndexedFiles

attachedIndexFiles

availableIndexFiles

FSE_FILE_SYSTEM_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

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

Pascal Structure

```
Defined in nwfse.inc

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

Fields

FATMovedCount

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

FATWriteErrorCount

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

someoneElseDidItCount0

Is used internally by the OS.

someoneElseDidItCount1

Is used internally by the OS.

someoneElseDidItCount2

Is used internally by the OS.

iRanOutSomeoneElseDidItCount0

Is used internally by the OS.

iRanOutSomeoneElseDidItCount1

Is used internally by the OS.

iRanOutSomeoneElseDidItCount2

Is used internally by the OS.

turboFATBuildScrewedUpCount

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

extraUseCountNodeCount

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

extraExtraUseCountNodeCount

errorReadingLastFATCount

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

someoneElseUsingThisFileCount

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

FSE_MM_OBJ_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

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

Pascal Structure

```
Defined in nwfse.inc

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

Fields

MediaInfo

mediaType

Specifies the media type of the object, as follows:

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

cartridgeType

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

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

unitSize

Specifies the current transfer unit size in bytes for the device.

blockSize

Specifies the size of a block for the device in bytes.

capacity

Specifies the capacity of the device in blocks.

preferredUnitSize

Specifies the preferred trans unit size for the device.

name

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

type

Specifies the media manager database type:

```
0   ADAPTER_OBJECT
```

1	CHANGER_OBJECT
2	RDEVICE_OBJECT
3	DEVICE_OBJECT
4	MDEVICE_OBJECT
4	RMEDIA_OBJECT
5	PARTITION_OBJECT
6	SLOT_OBJECT
7	HOTFIX_OBJECT
8	MIRROR_OBJECT
9	PARITY_OBJECT
10	VOLUME_SEG_OBJECT
11	VOLUME_OBJECT
12	CLONE_OBJECT
13	FMEDIA_OBJECT
14	UNKNOWN_OBJECT

status

Contains the status mask for the object:

OBJECT_ACTIVATED	0x00000001
OBJECT_CREATED	0x00000002
OBJECT_SCRAMBLED	0x00000004
OBJECT_RESERVED	0x00000010
OBJECT_BEING_IDENTIFIED	0x00000020
OBJECT_MAGAZINE_LOADED	0x00000040
OBJECT_FAILURE	0x00000080
OBJECT_REMOVABLE	0x00000100
OBJECT_READ_ONLY	0x00000200
OBJECT_IN_DEVICE	0x00010000
OBJECT_ACCEPTS_MAGAZIN	0x00020000
OBJECT_IS_IN_A_CHANGER	0x00040000
OBJECT_LOADABLE	0x00080000
OBJECT_BEING_LOADED	0x00080000
OBJECT_DEVICE_LOCK	0x01000000
OBJECT_CHANGER_LOCK	0x02000000
OBJECT_REMIRRORING	0x04000000
OBJECT_SELECTED	0x08000000

functionMask

Specifies the function mask:

0x0001	RANDOM_READ
0x0002	ANDOM_WRITE
0x0004	RANDOM_WRITE_ONCE
0x0008	SEQUENTIAL_READ
0x0010	SEQUENTIAL_WRITE
0x0020	RESET_END_OF_TAPE
0x0040	SINGLE_FILE_MARK
0x0080	MULTIPLE_FILE_MARK
0x0100	SINGLE_SET_MARK
0x0200	MULTIPLE_SET_MARK
0x0400	SPACE_DATA_BLOCKS
0x0800	LOCATE_DATA_BLOCKS

```
0x1000  POSITION_PARTITION
0x2000  POSITION_MEDIA
```

controlMask

Specifies the control mask:

```
0x0001  ACTIVATE_DEACTIVE
0x0002  MOUNT_DISMOUNT
0x0004  SELECT_UNSELECT
0x0008  LOCK_UNLOCK
0x0010  EJECT
0x0020  MOVE
```

parentCount

Specifies the number of parent objects for the device, usually 1.

siblingCount

Specifies the number of sibling objects for the device.

childCount

Specifies the number of child objects for the device.

specificInfoSize

Specifies the size of the data structures that will be returned.

objectUniqueID

Specifies the number which identifies the device in the media manager database.

mediaSlot

Specifies the number of the slot the device occupies.

FSE_SERVER_INFO

Returns information about the NetWare server

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   nuint32   replyCanceledCount;
   nuint32   writeHeldOffCount;
   nuint32   writeHeldOffWithDupRequest;
   nuint32   invalidRequestTypeCount;
   nuint32   beingAbortedCount;
   nuint32   alreadyDoingReallocCount;
   nuint32   deAllocInvalidSlotCount;
   nuint32   deAllocBeingProcessedCount;
   nuint32   deAllocForgedPacketCount;
   nuint32   deAllocStillTransmittingCount;
   nuint32   startStationErrorCount;
   nuint32   invalidSlotCount;
   nuint32   beingProcessedCount;
   nuint32   forgedPacketCount;
   nuint32   stillTransmittingCount;
   nuint32   reExecuteRequestCount;
   nuint32   invalidSequenceNumCount;
   nuint32   duplicateIsBeingSentAlreadyCnt;
   nuint32   sentPositiveAcknowledgeCount;
   nuint32   sentDuplicateReplyCount;
   nuint32   noMemForStationCtrlCount;
   nuint32   noAvailableConnsCount;
   nuint32   reallocSlotCount;
   nuint32   reallocSlotCameTooSoonCount;
} FSE_SERVER_INFO;
```

Pascal Structure

```
FSE_SERVER_INFO = Record
   replyCanceledCount : nuint32;
    writeHeldOffCount : nuint32;
    writeHeldOffWithDupRequest : nuint32;
    invalidRequestTypeCount : nuint32;
    beingAbortedCount : nuint32;
    alreadyDoingReallocCount : nuint32;
    deAllocInvalidSlotCount : nuint32;
    deAllocBeingProcessedCount : nuint32;
    deAllocForgedPacketCount : nuint32;
    deAllocStillTransmittingCount : nuint32;
    startStationErrorCount : nuint32;
```

```
      invalidSlotCount : nuint32;
      beingProcessedCount : nuint32;
      forgedPacketCount : nuint32;
      stillTransmittingCount : nuint32;
      reExecuteRequestCount : nuint32;
      invalidSequenceNumCount : nuint32;
      duplicateIsBeingSentAlreadyCnt : nuint32;
      sentPositiveAcknowledgeCount : nuint32;
      sentDuplicateReplyCount : nuint32;
      noMemForStationCtrlCount : nuint32;
      noAvailableConnsCount : nuint32;
      reallocSlotCount : nuint32;
      reallocSlotCameTooSoonCount : nuint32
    End;
```

Fields

replyCanceledCount

Specifies the number of replies that were cancelled because the connection was reallocated while the request was being processed.

writeHeldOffCount

Specifies the number of times that writes were delayed because of a pending TTS(tm) transaction or cache busy condition.

writeHeldOffWithDupRequest

Specifies the number of times that writes were cancelled since a duplicate request was received. (DO EITHER OF THESE REQUESTS GET WRITTEN? HOW ARE THEY PROCESSED--ORIGINAL OR DUPLICATE? HOW CAN THE GET THEM TO BE PROCESSED?)

invalidRequestTypeCount

Specifies the number of packets received which had an invalid request type or were received after the server was downed.

beingAbortedCount

Specifies the number of packets received for a connection being terminated.

alreadyDoingReallocCount

Specifies the number of times that a connection is requested when a connection already exists.

deAllocInvalidSlotCount

Specifies the number of times an attempt was made to deallocate a connection slot which was not valid.

deAllocBeingProcessedCount

Specifies the number of times the server was deallocated because requests were still being processed.

deAllocForgedPacketCount

Specifies the number of times the server was deallocated because a forget packet was received.

deAllocStillTransmittingCount

Specifies the number of times the server was deallocated because information was still being transmitted.

startStationErrorCount

Specifies the number of times the server was unable to allocate a connection for any reason.

invalidSlotCount

Specifies the number of requests received for an invalid connection slot.

beingProcessedCount

Specifies the number of times a duplicate request was received during processing of the first request.

forgedPacketCount

Specifies the number of suspicious invalid packets received.

stillTransmittingCount

Specifies the number of times a new request is received before a reply to a previous request has been sent.

reExecuteRequestCount

Specifies the number of times the requester did not receive the reply and the request had to be reprocessed.

invalidSequenceNumCount

Specifies the number of request packets the server received from a connection where the sequence number in the packet did not match the current sequence number or the next sequence number.

duplicateIsBeingSentAlreadyCnt

Specifies the number of times a duplicate reply was requested when the reply had already been sent.

sentPositiveAcknowledgeCount

Specifies the number of acknowledgments sent by the server (sent when a connection repeats a request being serviced).

sentDuplicateReplyCount

Specifies the number of request packets for which the server had to send a duplicate reply (only sent for requests the server cannot process).

noMemForStationCtrlCount

Specifies the number of times the server could not allocate memory to expand the connection table for a new connection.

noAvailableConnsCount

Specifies the number of times no slots were available in the connection table for a new connection.

reallocSlotCount

Specifies the number of times the server reallocated the same slot in the connection table for a client that logged out and relogged in.

reallocSlotCameTooSoonCount

Specifies the number of times that a request came from a client to relog in before that client had been completely logged out.

Remarks

It is rarely possible to create suspicious packets because of faulty equipment.

If the number specified by the *forgedPacketCount* and *invalidSequenceNumCount* fields are large, it may indicate an attempt to breach network security.

Packets with bad sequence numbers are discarded.

IPX_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   IPXSendPacketCount;
   nuint16   IPXMalformPacketCount;
   nuint32   IPXGetECBRequestCount;
   nuint32   IPXGetECBFailCount;
   nuint32   IPXAESEventCount;
   nuint16   IPXPostponedAESCount;
   nuint16   IPXMaxConfiguredSocketCount;
   nuint16   IPXMaxOpenSocketCount;
   nuint16   IPXOpenSocketFailCount;
   nuint32   IPXListenECBCount;
   nuint16   IPXECBCancelFailCount;
   nuint16   IPXGetLocalTargetFailCount;
} IPX_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  IPX_INFO = Record
    IPXSendPacketCount : nuint32;
    IPXMalformPacketCount : nuint16;
    IPXGetECBRequestCount : nuint32;
    IPXGetECBFailCount : nuint32;
    IPXAESEventCount : nuint32;
    IPXPostponedAESCount : nuint16;
    IPXMaxConfiguredSocketCount : nuint16;
    IPXMaxOpenSocketCount : nuint16;
    IPXOpenSocketFailCount : nuint16;
    IPXListenECBCount : nuint32;
    IPXECBCancelFailCount : nuint16;
    IPXGetLocalTargetFailCount : nuint16
  End;
```

Fields

IPXSendPacketCount

Specifies the number of IPX packets sent by the server.

IPXMalformPacketCount

Specifies the number of IPX packets discarded because they were malformed.

IPXGetECBRequestCount

Specifies the number of ECB requests.

IPXGetECBFailCount

Specifies the number of times an ECB was requested, but could not be supplied.

IPXAESEventCount

Specifies the number of AES events scheduled.

IPXPostponedAESCount

Specifies the number of AES events that could not be scheduled, but were placed in a waiting list.

IPXMaxConfiguredSocketCount

Specifies the maximum number of sockets that can be open at one time.

IPXMaxOpenSocketCount

Specifies the maximum number of sockets open at one time since the server was booted.

IPXOpenSocketFailCount

Specifies the number of times a request to open a socket failed.

IPXListenECBCount

Specifies the number of ECBs listening for a packet.

IPXECBCancelFailCount

Specifies the number of ECB listens that were cancelled.

IPXGetLocalTargetFailCount

Specifies the number of times the server failed to find the target.

KNOWN_NET_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   netIDNumber;
   nuint16   hopsToNet;
   nuint16   netStatus;
   nuint16   timeToNet;
} KNOWN_NET_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

KNOWN_NET_INFO = Record
    netIDNumber : nuint32;
    hopsToNet : nuint16;
    netStatus : nuint16;
    timeToNet : nuint16
  End;
```

Fields

netIDNumber

hopsToNet

netStatus

timeToNet

LAN_COMMON_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   notSupportedMask;
   nuint32   totalTxPacketCount;
   nuint32   totalRxPacketCount;
   nuint32   noECBAvailableCount;
   nuint32   packetTxTooBigCount;
   nuint32   packetTxTooSmallCount;
   nuint32   packetRxOverflowCount;
   nuint32   packetRxTooBigCount;
   nuint32   packetRxTooSmallCount;
   nuint32   packetTxMiscErrorCount;
   nuint32   packetRxMiscErrorCount;
   nuint32   retryTxCount;
   nuint32   checksumErrorCount;
   nuint32   hardwareRxMismatchCount;
   nuint32   reserved[50];
} LAN_COMMON_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  LAN_COMMON_INFO = Record
    notSupportedMask : nuint32;
    totalTxPacketCount : nuint32;
    totalRxPacketCount : nuint32;
    noECBAvailableCount : nuint32;
    packetTxTooBigCount : nuint32;
    packetTxTooSmallCount : nuint32;
    packetRxOverflowCount : nuint32;
    packetRxTooBigCount : nuint32;
    packetRxTooSmallCount : nuint32;
    packetTxMiscErrorCount : nuint32;
    packetRxMiscErrorCount : nuint32;
    retryTxCount : nuint32;
    checksumErrorCount : nuint32;
    hardwareRxMismatchCount : nuint32;
    reserved : Array[0..49] Of nuint32
  End;
```

Fields

Fields

notSupportedMask

Specifies a bit mask representing the fields in the statistics table. If the bit is 0, the counter is supported; if it is 1, the counter is not supported.

totalTxPacketCount

Specifies the total number of packets transmitted by the LAN board.

totalRxPacketCount

Specifies the total number of packets that were received by the LAN board.

noECBAvailableCount

Specifies the number of times the LAN board failed to get a receive ECB.

packetTxTooBigCount

Specifies the number of times the send packet was too big for this LAN board to send.

packetTxTooSmallCount

Specifies the number of times the send packet was too small for this LAN board to send.

packetRxOverflowCount

Specifies the number of times the LAN board's receive buffers overflowed.

packetRxTooBigCount

Specifies the number of times this LAN board could not receive a packet because the packet was too big.

packetRxTooSmallCount

Specifies the number of times this LAN board could not receive a packet because the packet was too small.

packetTxMiscErrorCount

Specifies the number of times any kind of transmit error occurred for this LAN board.

packetRxMiscErrorCount

Specifies the number of times any kind of receive error occurred for this LAN board.

retryTxCount

Specifies the number of times the LAN board retried a transmit because of failure.

checksumErrorCount

Specifies the number of times a checksum error occurred for this LAN board.

hardwareRxMismatchCount

Specifies a counter that may be incremented when a packet is received which does not pass length consistency checks.

reserved

LAN_CONFIG_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint8    DriverCFG_MajorVersion;
   nuint8    DriverCFG_MinorVersion;
   nuint8    DriverNodeAddress[6];
   nuint16   DriverModeFlags;
   nuint16   DriverBoardNum;
   nuint16   DriverBoardInstance;
   nuint32   DriverMaxSize;
   nuint32   DriverMaxRecvSize;
   nuint32   DriverRecvSize;
   nuint32   reserved1[3];
   nuint16   DriverCardID;
   nuint16   DriverMediaID;
   nuint16   DriverTransportTime;
   nuint8    DriverReserved[16];
   nuint8    DriverMajorVersion;
   nuint8    DriverMinorVersion;
   nuint16   DriverFlags;
   nuint16   DriverSendRetries;
   nuint32   DriverLink;
   nuint16   DriverSharingFlags;
   nuint16   DriverSlot;
   nuint16   DriverIOPortsAndLengths[4];
   nuint32   DriverMemDecode0;
   nuint16   DriverLength0;
   nuint32   DriverMemDecode1;
   nuint16   DriverLength1;
   nuint8    DriverInterrupt[2];
   nuint8    DriverDMAUsage[2];
   nuint32   Reserved2[3];
   nuint8    DriverLogicalName[18];
   nuint32   DriverLinearMem[2];
   nuint16   DriverChannelNum;
   nuint8    DriverIOReserved[6];
} LAN_CONFIG_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  LAN_CONFIG_INFO = Record
  DriverCFG_MajorVersion : nuint8;
  DriverCFG_MinorVersion : nuint8;
  DriverNodeAddress : Array[0..5] Of nuint8;
  DriverModeFlags : nuint16;
  DriverBoardNum : nuint16;
  DriverBoardInstance : nuint16;
  DriverMaxSize : nuint32;
  DriverMaxRecvSize : nuint32;
  DriverRecvSize : nuint32;
  Reserved1 : Array[0..2] Of nuint32;
  DriverCardID : nuint16;
  DriverMediaID : nuint16;
  DriverTransportTime : nuint16;
  DriverReserved : Array[0..15] Of nuint8;
  DriverMajorVersion : nuint8;
  DriverMinorVersion : nuint8;
  DriverFlags : nuint16;
  DriverSendRetries : nuint16;
  DriverLink : nuint32;
  DriverSharingFlags : nuint16;
  DriverSlot : nuint16;
  DriverIOPortsAndLengths : Array[0..3] Of nuint16;
  DriverMemDecode0 : nuint32;
  DriverLength0 : nuint16;
  DriverMemDecode1 : nuint32;
  DriverLength1 : nuint16;
  DriverInterrupt : Array[0..1] Of nuint8;
  DriverDMAUsage : Array[0..1] Of nuint8;
  Reserved2 : Array[0..2] Of nuint32;
  DriverLogicalName : Array[0..17] Of nuint8;
  DriverLinearMem : Array[0..1] Of nuint32;
  DriverChannelNum : nuint16;
  DriverIOReserved : Array[0..5] Of nuint8
End;
```

Fields

DriverCFG_MajorVersion

Specifies the Novell® defined major version number of the configuration table.

DriverCFG_MinorVersion

Specifies the Novell defined minor version of the configuration table.

DriverNodeAddress

Specifies the node address of the LAN board.

DriverModeFlags

Specifies the mode supported by the driver:

0x0001 Specifies whether the driver was real or a dummy; set to 1.
0x0002 Specifies if the driver uses DMA.

0x0004 Specifies to routers to pass router table changes when they occur, rather than forwarding all RIP and SAP packets; set only if but 4 is set.

0x0008 Specifies if the driver supports multicasting.

0x0010 Specifies if the driver can bind with a protocol stack without providing a network number.

0x0030 Specifies if the driver supports raw sends, no prepending any hardware header.

0x0400 Specifies if the HSM can handle fragmented RCBs.

0x2000 Specifies if the HSM can handle promiscuous RCBs.

0xC000 Specifies the driver node address, as follows:

```
00 Format is unspecified; the node address is           assumed to
01 Illegal combination
10 Driver node address is canonical
11 Driver node address is noncanonical
```

DriverBoardNum

Specifies the logical board number (1-255) assigned to the LAN board by the LSL(tm) service.

DriverBoardInstance

Specifies the number of the physical card the logical board is using.

DriverMaxSize

Specifies the maximum send or receive packet size in bytes the board can handle.

DriverMaxRecvSize

Specifies the maximum packet size in bytes that the LAN board can receive.

DriverRecvSize

Specifies the maximum packet size in bytes a protocol stack can send or receive using this board.

reserved1

DriverCardID

Specifies the number assigned to the LAN board by IMSP.

DriverMediaID

Specifies the number identifying the link-level envelope used by the MLID.

DriverTransportTime

Specifies the time in ticks to transmit a 576-byte packet.

DriverReserved

DriverMajorVersion

Specifies the major version number of the MLID.

DriverMinorVersion

DriverMinorVersion

Specifies the minor version number of the MLID.

DriverFlags

Specifies a bit map showing the architecture supported by the MLID:

```
0x0001   EISA
0x0002   ISA
0x0004   MCA
0x0100   Hub management
0x0600   Multicast filtering             and format:
         00 LAN medium defaults
         01 Illegal combination
```

The following bits are set if the board can share:

```
0x0020  Primary interrupt
0x0040  Secondary interrupt
0x0080  DMA channel 0
0x0100  DMA channel 1
```

The following bits are set if:

```
0x0200  A command line                  information string to
0x0400  To prevent default              information from the
```

DriverSendRetries

DriverLink

DriverSharingFlags

DriverSlot

Specifies the slot number of the board if installed in MCA or EISA machine; otherwise it is 0.

DriverIOPortsAndLengths

Each WORD is defined below:

```
Word 1  Primary base I/O port
Word 2  Number of I/O ports beginning with primary base I/O port
Word 3  Secondary base I/O port
Word 4  Number of I/O ports beginning with secondary base I/O port
```

DriverMemDecode0

Specifies the absolute primary memory address that the LAN board uses.

DriverLength0

Specifies the amount of memory in paragraphs the board uses starting at *DriverMemDecode0*

DriverMemDecode1

Specifies the absolute secondary memory address the board uses.

DriverLength1

Specifies the amount of memory in paragraphs the board uses, starting at *DriverMemDecode1*.

DriverInterrupt

Specifies the primary interrupt in the first byte; secondary interrupt in the secondary byte. FFh means not used.

DriverDMAUsage

Specifies the primary DMA channel used in the board in the first byte; secondary DMA channel in the second byte. FFh means not used.

Reserved2

Specifies the logical name of the LAN driver, given at load time.

DriverLogicalName

Specifies the logical name of the LAN driver, given at load time.

DriverLinearMem

Specifies the addresses of *DriverMemDecode0* and *DriverMemDecode1* in the first and second LONGS.

DriverChannelNum

Specifies the multichannel adapters. It holds the channel number of the NIC to use.

DriverIOReserved

LSL_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   rxBufs;
   nuint32   rxBufs75PerCent;
   nuint32   rxBufsCheckedOut;
   nuint32   rxBufMaxSize;
   nuint32   maxPhysicalSize;
   nuint32   lastTimeRxBufAllocated;
   nuint32   maxNumsOfProtocols;
   nuint32   maxNumsOfMediaTypes;
   nuint32   totalTXPackets;
   nuint32   getECBBfrs;
   nuint32   getECBFails;
   nuint32   AESEventCounts;
   nuint32   postponedEvents;
   nuint32   ECBCxlFails;
   nuint32   validBfrsReused;
   nuint32   enqueuedSendCount;
   nuint32   totalRXPackets;
   nuint32   unclaimedPackets;
   nuint8    StatisticsTableMajorVersion;
   nuint8    StatisticsTableMinorVersion;
] LSL_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 LSL_INFO = Record
    rxBufs : nuint32;
    rxBufs75PerCent : nuint32;
    rxBufsCheckedOut : nuint32;
    rxBufMaxSize : nuint32;
    maxPhysicalSize : nuint32;
    lastTimeRxBufAllocated : nuint32;
    maxNumsOfProtocols : nuint32;
    maxNumsOfMediaTypes : nuint32;
    totalTXPackets : nuint32;
    getECBBfrs : nuint32;
    getECBFails : nuint32;
    AESEventCounts : nuint32;
    postponedEvents : nuint32;
    ECBCxlFails : nuint32;
    validBfrsReused : nuint32;
    enqueuedSendCount : nuint32;
    totalRXPackets : nuint32;
    unclaimedPackets : nuint32;
    StatisticsTableMajorVersion : nuint8;
    StatisticsTableMinorVersion : nuint8
  End;
```

Fields

rxBufs

Specifies the total number of LSL receive buffers.

rxBufs75PerCent

Specifies the number of LSL receive buffers that must be in use before a warning message is issued that buffers are getting low.

rxBufsCheckedOut

Specifies the number of LSL buffers in use.

rxBufMaxSize

Specifies the size of the data portion of the ECBs in bytes.

maxPhysicalSize

Specifies the total size of the ECB in bytes.

lastTimeRxBufAllocated

Specifies the last time in ticks a buffer was checked out.

maxNumsOfProtocols

Specifies the number of protocol stacks supported by the OS.

maxNumsOfMediaTypes

Specifies the number of frame types supported by the OS.

totalTXPackets

Specifies the number of packet transmit requests. getECBBfrs contains the number of ECBs that were requested.

getECBBfrs

getECBFails

Specifies the number of times an ECB request failed.

AESEventCounts

Specifies the total number of AES events that have been processed.

postponedEvents

Specifies the total number of AES events postponed because of critical sections.

ECBCxlFails

Specifies the number of AES cancel requests that failed because the event was not found on the AES list.

validBfrsReused

Specifies the number of ECBs in the hold queue that were reused before they were removed from the hold queue.

enqueuedSendCount

Specifies the number of send events in the queue that have occurred.

totalRXPackets

Specifies the total number of received incoming packets.

unclaimedPackets

Specifies the total number of unclaimed incoming packets.

StatisticsTableMajorVersion

StatisticsTableMinorVersion

MEDIA_INFO_DEF

Returns information on the media manager object

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint8    label[64];
   nuint32   identificationType;
   nuint32   identificationTimeStamp;
} MEDIA_INFO_DEF;
```

Pascal Structure

```
Defined in nwfse.inc

 MEDIA_INFO_DEF = Record
    mediaLabel : Array[0..63] Of nuint8;
    identificationType : nuint32;
    identificationTimeStamp : nuint32
  End;
```

Fields

label

Specifies the name of the object.

identificationType

Specifies the Novell assigned number for the object.

identificationTimeStamp

Specifies the DOS timestamp of the object.

NLM_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   identificationNum;
   nuint32   flags;
   nuint32   type;
   nuint32   parentID;
   nuint32   majorVersion;
   nuint32   minorVersion;
   nuint32   revision;
   nuint32   year;
   nuint32   month;
   nuint32   day;
   nuint32   allocAvailableBytes;
   nuint32   allocFreeCount;
   nuint32   lastGarbageCollection;
   nuint32   messageLanguage;
   nuint32   numOfReferencedPublics;
} NLM_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NLM_INFO = Record
    identificationNum : nuint32;
    flags : nuint32;
    NLMtype : nuint32;
    parentID : nuint32;
    majorVersion : nuint32;
    minorVersion : nuint32;
    revision : nuint32;
    year : nuint32;
    month : nuint32;
    day : nuint32;
    allocAvailableBytes : nuint32;
    allocFreeCount : nuint32;
    lastGarbageCollection : nuint32;
    messageLanguage : nuint32;
    numOfReferencedPublics : nuint32
  End;
```

Fields

Fields

identificationNum

Specifies the number assigned to the NLM when it was loaded.

flags

Specifies a bit mask. Bits are defined as follows:

```
0x0000 = REENTRANT
0x0002 = MULTIPLE
0x0003 = SYNCHRONIZE
0x0008 = PSEUDOPREEMPTION
```

type

Specifies the type:

```
0 = NLM_GENERIC
1 = LAN_DRIVER
2 = DSK_DRIVER
3 = NAM_SPACE
4 = NLM_UTILITY
5 = MIRRORED_SERVER_LINK
6 = NLM_OS
7 = NLM_PAGED_HIGH_OS
8 = HOST_ADAPTER_MODULE
9 = CUSTOM_DEVICE_MODULE
10 = NLM_FILE_SYSTEM
11 = NLM_REAL_MODE
```

parentID

Specifies the number of the NLM that caused this NLM to be loaded.

majorVersion

Specifies the major version of the NLM.

minorVersion

Specifies the minor version of the NLM.

revision

Specifies the revision letter of the NLM.

year

Specifies the timestamp of the NLM.

month

Specifies the timestamp of the NLM.

day

Specifies the timestamp of the NLM.

allocAvailableBytes

Specifies the bytes available for allocation by the NLM.

allocFreeCount

Specifies the number of bytes freed that can be reclaimed.

lastGarbageCollection

Specifies the last time garbage collection was done for the NLM.

messageLanguage

Specifies the number representing the language the NLM uses.

numOfReferencedPublics

Specifies the number of external symbols referenced by the NLM.

NW_DYNAMIC_MEM

Service: Server Environment
Defined In: nwfserver.h

Structure

```
typedef struct
{
   nuint32   total;
   nuint32   max;
   nuint32   cur;
} NW_DYNAMIC_MEM;
```

Pascal Structure

```
Defined in nwserver.inc

 NW_DYNAMIC_MEM = Record
    total : nuint32;
     max : nuint32;
    cur : nuint32
 End;
```

Fields

total

Specifies the total amount of memory in dynamic memory area.

max

Specifies the amount of memory in dynamic memory area that has been in use since the server was brought up.

cur

Specifies the amount of memory in dynamic memory area currently in use.

NW_FS_INFO

Service: Server Environment
Defined In: nwfserver.inc

Structure

```
Defined          upTime;
   nuint8           processor;
   nuint8           reserved;
   nuint8           numProcs;
   nuint8           utilization;
   nuint16          configuredObjs;
   nuint16          maxObjs;
   nuint16          curObjs;
   nuint16          totalMem;
   nuint16          unusedMem;
   nuint16          numMemAreas;
   NW_DYNAMIC_MEM   dynamicMem[3];
} NW_FS_INFO;
```

Fields

upTime

Specifies how long the file server has been up in 1/18 ticks (wraps at 0xFFFFFFFF).

processor

1=8086/8088, 2=80286

reserved

numProcs

Specifies the number processes that handle incoming service requests.

utilization

Specifies the server utilization percentage (0 to 100), updated once per second.

configuredObjs

Specifies the maximum number of Bindery objects the file server will track. A 0 means an unlimited number, and *maxObjs* and *curObjs* have no meaning.

maxObjs

Specifies the maximum number of Bindery objects that have been used concurrently since the file server came up.

curObjs

Specifies the actual number of Bindery objects currently in use on the server.

totalMem

Specifies the total amount of memory installed on the server.

unusedMem

Specifies the amount of memory the server has determined it is not using.

numMemAreas

Specifies the number of dynamic memory areas (1 to 3).

dynamicMem

NWFSE_ACTIVE_CONN_LIST

Returns the Active Connection List by type

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   nuint8                    activeConnBitList[512];
} NWFSE_ACTIVE_CONN_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_ACTIVE_CONN_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    activeConnBitList : Array[0..511] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

activeConnBitList

NWFSE_ACTIVE_LAN_BOARD_LIST

Returns the active LAN board list
Service: Server Environment
Defined In: nwfserver.inc

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   nuint32                   MaxNumOfLANs;
   nuint32                   LANLoadedCount;
   nuint32                   boardNums[FSE_MAX_NUM_OF_LANS];
} NWFSE_ACTIVE_LAN_BOARD_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_ACTIVE_LAN_BOARD_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    MaxNumOfLANs : nuint32;
    LANLoadedCount : nuint32;
    boardNums : Array[0..FSE_MAX_NUM_OF_LANS-1] Of nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

MaxNumOfLANs

MaxNumOfLANs

boardNums

NWFSE_ACTIVE_STACKS

Returns information about active protocol stacks

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    maxNumOfStacks;
   nuint32                    stackCount;
   nuint32                    nextStartNum;
   STACK_INFO                 stackInfo[FSE_MAX_NUM_OF_STACKINFO];
} NWFSE_ACTIVE_STACKS;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_ACTIVE_STACKS = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    maxNumOfStacks : nuint32;
    stackCount : nuint32;
    nextStartNum : nuint32;
    stackInfo : Array[0.. FSE_MAX_NUM_OF_STACKINFO -1] Of STACK_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

maxNumOfStacks

Specifies the total number of protocol stacks.

stackCount

Specifies the number of STACK_INFO structures in the buffer.

nextStartNum

stackInfo

NWFSE_CACHE_INFO

Returns Server Environment cache information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   CACHE_COUNTERS             cacheCounters;
   CACHE_MEM_COUNTERS         cacheMemCounters;
   CACHE_TREND_COUNTERS       cacheTrendCounters;
   CACHE_INFO                 cacheInformation;
} NWFSE_CACHE_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_CACHE_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    cacheCounters : CACHE_COUNTERS;
    cacheMemCounters : CACHE_MEM_COUNTERS;
    cacheTrendCounters : CACHE_TREND_COUNTERS;
    cacheInformation : CACHE_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to SERVER_AND_VCONSOLE_INFO.

reserved

Is reserved for future use.

cacheCounters

Points to CACHE_COUNTERS.

cacheMemCounters

Points to CACHE_MEM_COUNTERS.

cacheTrendCounters

Points to CACHE_TREND_COUNTERS.

cacheInformation

Points to CACHE_INFO.

NWFSE_CPU_INFO

Returns Server Environment CPU information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   nuint32                   numOfCPUs;
   CPU_INFO                  CPUInfo;
} NWFSE_CPU_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_CPU_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numOfCPUs : nuint32;
    CPUInfo : CPU_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to SERVER_AND_VCONSOLE_INFO.

reserved

Is reserved (pass 0).

numOfCPUs

Specifies the number of CPUs in the server.

CPUInfo

Points to the CPU_INFO structure.

NWFSE_DIR_CACHE_INFO

Returns Directory cache information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   DIR_CACHE_INFO             dirCacheInfo;
} NWFSE_DIR_CACHE_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_DIR_CACHE_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    dirCacheInfo : DIR_CACHE_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

dirCacheInfo

NWFSE_FILE_SERVER_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    NCPStationsInUseCount;
   nuint32                    NCPPeakStationsInUseCount;
   nuint32                    numOfNCPRequests;
   nuint32                    serverUtilization;
   FSE_SERVER_INFO            ServerInfo;
   FILE_SERVER_COUNTERS       fileServerCounters;
} NWFSE_FILE_SERVER_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_FILE_SERVER_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    NCPStationsInUseCount : nuint32;
    NCPPeakStationsInUseCount : nuint32;
    numOfNCPRequests : nuint32;
    serverUtilization : nuint32;
    ServerInfo : FSE_SERVER_INFO;
    fileServerCounters : FILE_SERVER_COUNTERS
  End;
```

Fields

serverTimeAndVConsoleInfo

Specifies the time elapsed since the server was brought up (returned in ticks---about 1/18 seconds) and the console version and revision numbers to track packet information. When *serverTimeAndVConsoleInfo* reaches 0xFFFFFFFF, it wraps to zero.

reserved

NCPStationsInUseCount

Specifies the number of workstations connected to the server.

NCPPeakStationsInUseCount

Specifies the maximum number of workstations connected at one time since the server was brought up.

numOfNCPRequests

Specifies the number of NCP requests received by the server since it was brought up.

serverUtilization

Specifies the current percentage of CPU utilization for the server.

ServerInfo

Specifies the NetWare server statistics.

fileServerCounters

Specifies the NetWare server statistics.

NWFSE_FILE_SYSTEM_INFO

Returns NetWare File Systems information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   FSE_FILE_SYSTEM_INFO       fileSystemInfo;
} NWFSE_FILE_SYSTEM_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_FILE_SYSTEM_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    fileSystemInfo : FSE_FILE_SYSTEM_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Specifies the time elapsed since the server was brought up (returned in ticks---about 1/18 seconds) and the console version and revision numbers to track packet information. When *serverTimeAndVConsoleInfo* reaches 0xFFFFFFFF, it wraps to zero.

reserved

fileSystemInfo

NWFSE_GARBAGE_COLLECTION_INFO

Returns information about failed requests

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    failedAllocRequestCount;
   nuint32                    numOfAllocs;
   nuint32                    noMoreMemAvailableCount;
   nuint32                    numOfGarbageCollections;
   nuint32                    garbageFoundSomeMem;
   nuint32                    garbageNumOfChecks;
} NWFSE_GARBAGE_COLLECTION_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_GARBAGE_COLLECTION_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    failedAllocRequestCount : nuint32;
    numOfAllocs : nuint32;
    noMoreMemAvailableCount : nuint32;
    numOfGarbageCollections : nuint32;
    garbageFoundSomeMem : nuint32;
    garbageNumOfChecks : nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass zero).

failedAllocRequestCount

Specifies the number of memory allocations that failed since the server was brought up.

numOfAllocs

Specifies the number of memory allocations made since the server was

Specifies the number of memory allocations made since the server was brought up.

noMoreMemAvailableCount

Specifies the number of times that allocation failed because there was no memory available since the server was brought up.

numOfGarbageCollections

Specifies the number of times garbage collection was invoked since the server was brought up.

garbageFoundSomeMem

Specifies the number of times garbage collection reclaimed memory since the server was brought up.

garbageNumOfChecks

Specifies the number of times garbage collection checked for memory since the server was brought up.

NWFSE_GENERAL_ROUTER_SAP_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo
   nuint16                    reserved;
   nuint32                    internalRIPSocket;
   nuint32                    internalRouterDownFlag;
   nuint32                    trackOnFlag;
   nuint32                    externalRouterActiveFlag;
   nuint32                    internalSAPSocketNumber;
   nuint32                    replyToNearestServerFlag;
} NWFSE_GENERAL_ROUTER_SAP_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_GENERAL_ROUTER_SAP_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    internalRIPSocket : nuint32;
    internalRouterDownFlag : nuint32;
    trackOnFlag : nuint32;
    externalRouterActiveFlag : nuint32;
    internalSAPSocketNumber : nuint32;
    replyToNearestServerFlag : nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

internalRIPSocket

Specifies the router socket number.

internalRouterDownFlag

Specifies whether the internal router is up or down.

trackOnFlag

Specifies whether router tracking is active (the console operator issued the TRACK ON console command).

externalRouterActiveFlag

Specifies whether an external router is active.

internalSAPSocketNumber

Specifies the number of the socket that receives SAP packets.

replyToNearestServerFlag

Specifies whether the server will respond to **GetNearestServer**.

NWFSE_IPXSPX_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   IPX_INFO                  IPXInfo;
   SPX_INFO                  SPXInfo;
} NWFSE_IPXSPX_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_IPXSPX_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    IPXInfo : IPX_INFO;
    SPXInfo : SPX_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

IPXInfo

SPXInfo

NWFSE_KNOWN_NETWORKS_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo
   nuint16                   reserved;
   nuint32                   numberOfEntries;
   KNOWN_NET_INFO            knownNetInfo[51];
} NWFSE_KNOWN_NETWORKS_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_KNOWN_NETWORKS_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numberOfEntries : nuint32;
    knownNetInfo : Array[0..50] Of KNOWN_NET_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Specifies the time elapsed since the server was brought up, and the console version number.

reserved

numberOfEntries

knownNetInfo

Specifies the first *knownNetworkStructure*.

NWFSE_KNOWN_SERVER_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    numberOfEntries;
   nuint8                     data[512];
} NWFSE_KNOWN_SERVER_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_KNOWN_SERVER_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numberOfEntries : nuint32;
    data : Array[0..511] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass 0).

numberOfEntries

Specifies the number of entries.

data

Specifies an array containing the following fields:

```
SERVER_INFO=RECORD    network Address:
Array[0..3] of BYTE;    nodeAddress:
Array[0..5] of BYTE;    socketAddress:
nuint16;    HopsToServer:       nuint16;
ServerName:          Array[0..47] of char8;  (#0
terminated) END;
```

The next field starts immediately after the trailing #0 of the last *ServerName* field.

NWFSE_LAN_COMMON_COUNTERS_INFO

Returns information on LAN common counters

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint8                     statisticsMajorVersion;
   nuint8                     statisticsMinorVersion;
   nuint32                    numberOfGenericCounters;
   nuint32                    numberOfCounterBlocks;
   nuint32                    customVariableCount;
   nuint32                    NextCounterBlock;
   LAN_COMMON_INFO            LANCommonInfo;
} NWFSE_LAN_COMMON_COUNTERS_INFO;
```

Pascal Structure

```
NWFSE_LAN_COMMON_COUNTERS_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    statisticsMajorVersion : nuint8;
    statisticsMinorVersion : nuint8;
    numberOfGenericCounters : nuint32;
    numberOfCounterBlocks : nuint32;
    customVariableCount : nuint32;
    NextCounterBlock : nuint32;
    LANCommonInfo : LAN_COMMON_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

statisticsMajorVersion

Specifies the major version number of the statistics table.

statisticsMinorVersion

Specifies the minor version number of the statistics table.

numberOfGenericCounters

Specifies the total number of LAN common counters.

numberOfCounterBlocks

Specifies the number of blocks used by LAN common counters by the

LAN board.

customVariableCount

Specifies the number of custom counters for this LAN board.

NextCounterBlock

Specifies the value to be passed in block numbers to the **NWGetLANCommonCountersInfo** function.

LANCommonInfo

Points to the LAN_COMMON_INFO structure containing information about the LAN board.

Remarks

When 0 is returned in the *NextCounterBlock* field, all common counters have been returned.

NWFSE_LAN_CONFIG_INFO

Returns LAN configuration information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   LAN_CONFIG_INFO            LANConfigInfo;
} NWFSE_LAN_CONFIG_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_LAN_CONFIG_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    LANConfigInfo : LAN_CONFIG_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass 0).

LANConfigInfo

NWFSE_LAN_CUSTOM_INFO

Returns information on LAN custom counters

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    numCustomVar;
   nuint8                     customInfo[512];
} NWFSE_LAN_CUSTOM_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_LAN_CUSTOM_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numCustomVar : nuint32;
    customInfo : Array[0..511] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass 0).

numCustomVar

Specifies the value of the custom counter.

customInfo

Specifies the description of the custom counter.

NWFSE_LOADED_MEDIA_NUM_LIST

Returns a list of loaded media numbers
Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    maxMediaTypes;
   nuint32                    mediaListCount;
   nuint32                    mediaList[FSE_MEDIA_LIST_MAX];
} NWFSE_LOADED_MEDIA_NUM_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_LOADED_MEDIA_NUM_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    maxMediaTypes : nuint32;
    mediaListCount : nuint32;
    mediaList : Array[0.. FSE_MEDIA_LIST_MAX -1] Of nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass 0).

maxMediaTypes

Specifies the maximum number of media allowed.

mediaListCount

Specifies the number of valid IDs returned in the *mediaList* parameter.

mediaList

Specifies the ID numbers of the returned media.

NWFSE_LSL_INFO

Returns LSL information

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   LSL_INFO                   LSLInfo;
} NWFSE_LSL_INFO;
```

Pascal Structure

```
NWFSE_LSL_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    LSLInfo : LSL_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass 0).

LSLInfo

Points to the LSL_INFO structure containing the LSL information.

NWFSE_LSL_LOGICAL_BOARD_STATS

Returns statistics concerning LSL boards

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved0;
   nuint32                    LogTtlTxPackets;
   nuint32                    LogTtlRxPackets;
   nuint32                    LogUnclaimedPackets;
   nuint32                    reserved1;
} NWFSE_LSL_LOGICAL_BOARD_STATS;
```

Pascal Structure

```
NWFSE_LSL_LOGICAL_BOARD_STATS = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved0 : nuint16;
    LogTtlTxPackets : nuint32;
    LogTtlRxPackets : nuint32;
    LogUnclaimedPackets : nuint32;
    reserved1 : nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved0

Is reserved (pass zero).

LogTtlTxPackets

Specifies the total number of packets transmitted.

LogTtlRxPackets

Specifies the total number of packets received.

LogUnclaimedPackets

Specifies the total number of unclaimed packets.

reserved1

Is reserved (pass zero).

NWFSE_MEDIA_MGR_OBJ_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   FSE_MM_OBJ_INFO            fseMMObjInfo;
} NWFSE_MEDIA_MGR_OBJ_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_MEDIA_MGR_OBJ_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    fseMMObjInfo : FSE_MM_OBJ_INFO
  End;
```

Fields

reserved

fseMMObjInfo

NWFSE_MEDIA_MGR_OBJ_LIST

Returns the media manager object list and the media manager object children's list

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    nextStartObjNum;
   nuint32                    objCount;
   nuint32                    objs[FSE_MAX_OBJECTS];
} NWFSE_MEDIA_MGR_OBJ_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_MEDIA_MGR_OBJ_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    nextStartObjNum : nuint32;
    objCount : nuint32;
    objs : Array[0.. FSE_MAX_OBJECTS -1] Of nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

nextStartObjNum

objCount

Specifies the number of object IDs returned.

objs

Specifies the list of object IDs.

NWFSE_MEDIA_NAME_LIST

Returns the media name by using a media number

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
} NWFSE_MEDIA_NAME_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_MEDIA_NAME_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

NWFSE_NETWORK_ROUTER_INFO

Returns information about a specified router on the network

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    NetIDNumber;
   nuint16                    HopsToNet;
   nuint16                    NetStatus;
   nuint16                    TimeToNet;
} NWFSE_NETWORK_ROUTER_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_NETWORK_ROUTER_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    NetIDNumber : nuint32;
    HopsToNet : nuint16;
    NetStatus : nuint16;
    TimeToNet : nuint16
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass zero).

NetIDNumber

Specifies the network ID number used by the server.

HopsToNet

Specifies the number of routers to cross to get to the network.

NetStatus

Specifies the status of the network.

TimeToNet

Specifies the number of clock ticks to the network (roundtrip).

Remarks

The *NetStatus* field can have the following values:

0x01	LOCALBIT
0x02	NETSTARTBIT
0x04	NETRELIABLEBIT
0x10	NETWANBIT

NWFSE_NETWORK_ROUTERS_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    NumberOfEntries;
   ROUTERS_INFO               routersInfo[36];
} NWFSE_NETWORK_ROUTERS_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_NETWORK_ROUTERS_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    NumberOfEntries : nuint32;
    routersInfo : Array[0..35] Of ROUTERS_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

NumberOfEntries

routersInfo

NWFSE_NLM_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   NLM_INFO                  NLMInfo;
} NWFSE_NLM_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_NLM_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    NLMInfo : NLM_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

NLMInfo

NWFSE_NLM_LOADED_LIST

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   nuint32                   numberNLMsLoaded;
   nuint32                   NLMsInList;
   nuint32                   NLMNums[FSE_NLM_NUMS_RETURNED_MAX];
} NWFSE_NLM_LOADED_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_NLM_LOADED_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numberNLMsLoaded : nuint32;
    NLMsInList : nuint32;
    NLMNums : Array[0..FSE_NLM_NUMS_RETURNED_MAX-1] Of nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

numberNLMsLoaded

Specifies the total number of NLMs loaded on the server including hidden NLMs. No information will be returned about hidden NLMs.

NLMsInList

Specifies the number of valid NLM IDs returned in *NLMNums*. A valid NLM is an NLM whose information was placed in the buffer and does not include hidden NLMs.

NLMNums

NWFSE_NLMS_RESOURCE_TAG_LIST

Returns the NLM's resource tag list

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO  serverTimeAndVConsoleInfo;
   nuint16                   reserved;
   nuint32                   totalNumOfResourceTags;
   nuint32                   packetResourceTags;
   nuint8                    resourceTagBuf[512];
} NWFSE_NLMS_RESOURCE_TAG_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_NLMS_RESOURCE_TAG_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    totalNumOfResourceTags : nuint32;
    packetResourceTags : nuint32;
    resourceTagBuf : Array[0..511] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

totalNumOfResourceTags

Specifies the total number of resource tags the NLM is using.

packetResourceTags

Specifies the number of resource tags the structure contains.

resourceTagBuf

Contains the resourceTagBuf structure.

NWFSE_OS_VERSION_INFO

Returns Operating System version information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint8                     OSMajorVersion;
   nuint8                     OSMinorVersion;
   nuint8                     OSRevisionNum;
   nuint8                     accountingVersion;
   nuint8                     VAPVersion;
   nuint8                     queueingVersion;
   nuint8                     securityRestrictionsLevel;
   nuint8                     bridgingSupport;
   nuint32                    maxNumOfVolumes;
   nuint32                    numOfConnSlots;
   nuint32                    maxLoggedInConns;
   nuint32                    maxNumOfNameSpaces;
   nuint32                    maxNumOfLans;
   nuint32                    maxNumOfMediaTypes;
   nuint32                    maxNumOfProtocols;
   nuint32                    maxMaxSubdirTreeDepth;
   nuint32                    maxNumOfDataStreams;
   nuint32                    maxNumOfSpoolPrinters;
   nuint32                    serialNum;
   nuint16                    applicationNum;
} NWFSE_OS_VERSION_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_OS_VERSION_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    OSMajorVersion : nuint8;
    OSMinorVersion : nuint8;
    OSRevisionNum : nuint8;
    accountingVersion : nuint8;
    VAPVersion : nuint8;
    queueingVersion : nuint8;
    securityRestrictionsLevel : nuint8;
    bridgingSupport : nuint8;
```

```
     maxNumOfVolumes : nuint32;
     numOfConnSlots : nuint32;
     maxLoggedInConns : nuint32;
     maxNumOfNameSpaces : nuint32;
     maxNumOfLans : nuint32;
     maxNumOfMediaTypes : nuint32;
     maxNumOfProtocols : nuint32;
     maxMaxSubdirTreeDepth : nuint32;
     maxNumOfDataStreams : nuint32;
     maxNumOfSpoolPrinters : nuint32;
     serialNum : nuint32;
     applicationNum : nuint16
   End;
```

Fields

serverTimeAndVConsoleInfo

reserved

OSMajorVersion

Specifies the major version number of the OS.

OSMinorVersion

Specifies the minor version number of the OS.

OSRevisionNum

Specifies the version revision letter of the OS.

accountingVersion

Specifies the version of the accounting subsystem.

VAPVersion

Is unused.

queueingVersion

Specifies the queueing version number.

securityRestrictionsLevel

Specifies the security restriction version number.

bridgingSupport

Specifies the internet bridge support version number.

maxNumOfVolumes

numOfConnSlots

Specifies the maximum number of connections that can be used simultaneously on the server.

maxLoggedInConns

maxNumOfNameSpaces

Specifies the maximum number of name spaces that can be simultaneously loaded on the server.

maxNumOfLans

Specifies the maximum number of LAN boards that can be used on the server.

maxNumOfMediaTypes

Specifies the maximum number of different media types allowed on the server.

maxNumOfProtocols

Specifies the maximum number of protocol stacks that can be used on the server.

maxMaxSubdirTreeDepth

Specifies the maximum depth of directories that can be used on the server.

maxNumOfDataStreams

Specifies the maximum number of data streams that can be used on the server.

maxNumOfSpoolPrinters

Specifies the maximum number of spool printers (default queue assignments) that can be used on the server.

serialNum

Specifies the serial number of the server.

applicationNum

Is included for backward compatibility.

NWFSE_PACKET_BURST_INFO

Returns packet burst information

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO    serverTimeAndVConsoleInfo;
   nuint16                     reserved;
   PACKET_BURST_INFO           packetBurstInfo;
} NWFSE_PACKET_BURST_INFO;
```

Pascal Structure

```
NWFSE_PACKET_BURST_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    packetBurstInfo : PACKET_BURST_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass zero).

packetBurstInfo

Points to the PACKET_BURST_INFO structure containing information about packet bursts.

NWFSE_PROTOCOL_CUSTOM_INFO

Returns custom information about protocol stacks

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved0;
   nuint32                    customCount;
   nuint8                     customStruct[512];
} NWFSE_PROTOCOL_CUSTOM_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_PROTOCOL_CUSTOM_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved0 : nuint16;
    customCount : nuint32;
    customStruct : Array[0.. 512 -1] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved0

customCount

customStruct

Specifies the structure with the DWORD value of the custom counter, followed by a length preceded string describing the custom counter.

NWFSE_PROTOCOL_ID_NUMS

Returns the protocol stack numbers using a media number or using a LAN board number

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    stackIDCount;
   nuint32                    stackIDs[FSE_STACK_IDS_MAX];
} NWFSE_PROTOCOL_ID_NUMS;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_PROTOCOL_ID_NUMS = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    stackIDCount : nuint32;
    stackIDs : Array[0.. FSE_STACK_IDS_MAX -1] Of nuint32
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

stackIDCount

Specifies the number of protocol stack ID Number returned in *stackIDs* by **NWGetProtocolStkNumsByMediaNum**

stackIDs

Specifies the list of stack ID numbers.

NWFSE_PROTOCOL_STK_CONFIG_INFO

Returns information about the protocol stack configuration
Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint8                     configMajorVersionNum;
   nuint8                     configMinorVersionNum;
   nuint8                     stackMajorVersionNum;
   nuint8                     stackMinorVersionNum;
   nuint8                     stackShortName[16];
} NWFSE_PROTOCOL_STK_CONFIG_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_PROTOCOL_STK_CONFIG_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    configMajorVersionNum : nuint8;
    configMinorVersionNum : nuint8;
    stackMajorVersionNum : nuint8;
    stackMinorVersionNum : nuint8;
    stackShortName : Array[0..15] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

configMajorVersionNum
Specifies the major version number of the configuration table.

configMinorVersionNum
Specifies the minor version number of the configuration table.

stackMajorVersionNum
Specifies the major version number of the protocol stack.

stackMinorVersionNum

Specifies the minor version number of the protocol stack.

stackShortName

Specifies the short protocol name; name used to register the stack with the LSL. The first byte is the length of the string followed by the string itself. It is not a null-terminated string.

NWFSE_PROTOCOL_STK_STATS_INFO

Returns information about protocol stack statistics

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint8                     statMajorVersionNum;
   nuint8                     statMinorVersionNum;
   nuint16                    commonCounters;
   nuint32                    validCountersMask;
   nuint32                    totalTxPackets;
   nuint32                    totalRxPackets;
   nuint32                    ignoredRxPackets;
   nuint16                    numCustomCounters;
} NWFSE_PROTOCOL_STK_STATS_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_PROTOCOL_STK_STATS_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    statMajorVersionNum : nuint8;
    statMinorVersionNum : nuint8;
    commonCounters : nuint16;           (* always set to 3? *)
    validCountersMask : nuint32;
    totalTxPackets : nuint32;
    totalRxPackets : nuint32;
    ignoredRxPackets : nuint32;
    numCustomCounters : nuint16
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

statMajorVersionNum

Specifies the major version number of the statistics table.

statMinorVersionNum

Specifies the minor version number of the statistics table.

commonCounters

Specifies the number of counters in the fixed portion of the table, current 3.

validCountersMask

Specifies which counters are valid, right most bit for the first counter; 0 = Counter is valid, 1 = Counter is invalid

totalTxPackets

totalRxPackets

Specifies the total number of packets that were requested to be transmitted.

ignoredRxPackets

Specifies the number of incoming packets that were ignored by the stack.

numCustomCounters

Specifies the number of custom counters for the protocol stack.

NWFSE_SERVER_INFO

Returns server information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint8                     serverAddress[12];
   nuint16                    hopsToServer;
} NWFSE_SERVER_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

NWFSE_SERVER_INFO = Record
   serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
   reserved : nuint16;
   serverAddress : Array[0..11] Of nuint8;
   hopsToServer : nuint16
End;
```

Fields

serverTimeAndVConsoleInfo

reserved

serverAddress

hopsToServer

NWFSE_SERVER_SET_CATEGORIES

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo
   nuint16                    reserved;
   nuint32                    numberOfSetCategories;
   nuint32                    nextSequenceNumber;
   nuint8                     categoryName[512];
} NWFSE_SERVER_SET_CATEGORIES;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_SERVER_SET_CATEGORIES = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numberOfSetCategories : nuint32;
    nextSequenceNumber : nuint32;
    categoryName : Array[0..511] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

numberOfSetCategories

Specifies the total number of set categories supported on the server.

nextSequenceNumber

Specifies the number to be used for *startNum* on subsequent calls.

categoryName

Specifies the null-terminated (not length-preceded) string describing the category.

NWFSE_SERVER_SET_CMDS_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    numberOfSetCommands;
   nuint32                    nextSequenceNumber;
   nuint32                    setCmdType;
   nuint32                    setCmdCategory;
   nuint32                    setCmdFlags;
   nuint8                     setNameAndValueInfo[500];
} NWFSE_SERVER_SET_CMDS_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_SERVER_SET_CMDS_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numberOfSetCommands : nuint32;
    nextSequenceNumber : nuint32;
    setCmdType : nuint32;
    setCmdCategory : nuint32;
    setCmdFlags : nuint32;
    setNameAndValueInfo : Array[0..499] Of nuint8
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

numberOfSetCommands

Specifies the total number of set commands for all the categories on the server.

nextSequenceNumber

Specifies the next number to be used for *startNum* on the next call.

setCmdType

Specifies how to interpret the command as follows:

```
0   FSE_TYPE_NUMBER
1   FSE_TYPE_BOOLEAN
2   FSE_TYPE_TICKS
3   FSE_TYPE_BLOCK_SHIFT (512*number)
4   FSE_TYPE_TIME_OFFSET ([+|-]hh:mm:ss converted to seconds )
5   FSE_TYPE_STRING
6   FSE_TYPE_TRIGGER
```

The following show the types of triggers:

```
0x00   FSE_TYPE_TRIGGER_OFF
0x01   FSE_TYPE_TRIGGER_ON
0x10   FSE_TYPE_TRIGGER_PENDING
0x20   FSE_TYPE_TRIGGER_SUCCESS
0x30   FSE_TYPE_TRIGGER_FAILED
```

setCmdCategory

Specifies the category the command belongs to, as follows:

```
0   FSE_COMMUNICATIONS
1   FSE_MEMORY
2   FSE_FILE_CACHE
3   FSE_DIR_CACHE
4   FSE_FILE_SYSTEM
5   FSE_LOCKS
6   FSE_TRANSACTION_TRACKING
7   FSE_DISK
8   FSE_TIME
9   FSE_NCP
10  FSE_MISCELLANEOUS
11  FSE_ERRORS
```

setCmdFlags

Specifies the ways in which this category may be changed, as follows:

```
0x01   FSE_STARTUP_ONLY
0x02   FSE_HIDE
0x04   FSE_ADVANCED
0x08   FSE_STARTUP_OR_LATER
0x10   FSE_NOT_SECURED_CONSOLE (Can't be performed on secured conso
```

setNameAndValueInfo

Specifies the null-terminated string containing the name of the command at index 0, and the value that begins at strlen(name). The string is not length-preceded.

NWFSE_SERVER_SRC_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    numberOfEntries;
   SERVERS_SRC_INFO           serverSrcInfo[42];
} NWFSE_SERVER_SRC_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 NWFSE_SERVER_SRC_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numberOfEntries : nuint32;
    serversSrcInfo : Array[0..41] Of SERVERS_SRC_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

numberOfEntries

Specifies the number of SERVERS_SRC_INFOs that were returned by **NWGetServerSourcesInfo**.

serverSrcInfo

Specifies SERVERS_SRC_INFO.

NWFSE_USER_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   USER_INFO                  userInfo;
} NWFSE_USER_INFO;
```

Pascal Structure

```
 NWFSE_USER_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    userInfo : USER_INFO
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

userInfo

NWFSE_VOLUME_INFO_BY_LEVEL

Returns volume information when the information level is specified

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    infoLevel;
   VOLUME_INFO_BY_LEVEL       volumeInfo;
} NWFSE_VOLUME_INFO_BY_LEVEL;
```

Pascal Structure

```
NWFSE_VOLUME_INFO_BY_LEVEL = Record
    serverAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    infoLevel : nuint32;
    volumeInfo : VOLUME_INFO_BY_LEVEL
  End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass zero).

infoLevel

Specifies the information level to be returned.

volumeInfo

Points to the VOLUME_INFO_BY_LEVEL structure containing the specified volume information.

Remarks

If the *infoLevel* field is set to 1, the *volInfoDef* field of the VOLUME_INFO_BY_LEVEL structure will be used.

If the *infoLevel* field is set to 2, the *volInfoDef*2 field of the VOLUME_INFO_BY_LEVEL structure will be used.

NWFSE_VOLUME_SEGMENT_LIST

Returns the volume segment list

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    numOfVolumeSegments;
   VOLUME_SEGMENT             volumeSegment[42];
} NWFSE_VOLUME_SEGMENT_LIST;
```

Pascal Structure

```
Defined in nwfse.inc

  NWFSE_VOLUME_SEGMENT_LIST = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    numOfVolumeSegments : nuint32;
    volumeSegment : Array[0..41] Of VOLUME_SEGMENT (
  End;
```

Fields

serverTimeAndVConsoleInfo

reserved

numOfVolumeSegments

Specifies the number of volume segments on the volume.

volumeSegment

Specifies the volume information structures for all the volume segments on the volume. Only the number of structures will contain valid data.

NWFSE_VOLUME_SWITCH_INFO

Returns information about volume switches

Service: Server Environment

Defined In: nwfse.h and nwfse.inc

Structure

```
typedef struct {
   SERVER_AND_VCONSOLE_INFO   serverTimeAndVConsoleInfo;
   nuint16                    reserved;
   nuint32                    totalLFSCounters;
   nuint32                    CurrentLFSCounters;
   nuint32                    LFSCounters[128];
} NWFSE_VOLUME_SWITCH_INFO;
```

Pascal Structure

```
NWFSE_VOLUME_SWITCH_INFO = Record
    serverTimeAndVConsoleInfo : SERVER_AND_VCONSOLE_INFO;
    reserved : nuint16;
    totalLFSCounters : nuint32;
    CurrentLFSCounters : nuint32;
    LFSCounters : Array[0..127] Of nuint32 (* 512 / sizeof(nuint32) *)
End;
```

Fields

serverTimeAndVConsoleInfo

Points to the SERVER_AND_VCONSOLE_INFO structure containing the time since the server was brought up.

reserved

Is reserved (pass zero).

totalLFSCounters

Specifies the total number of FS counters.

CurrentLFSCounters

Specifies the current number of FS counters.

LFSCounters

Specifies the number of LFS counters.

NWLAN_CONFIG

Returns configuration data for a LAN card on a NetWare server

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint8   networdAddress[4];
   nuint8   hostAddress[6];
   nuint8   boardInstalled;
   nuint8   optionNumber;
   nuint8   configurationText1[80];
   nuint8   configurationText2
```

Pascal Structure

```
Defined in nwserver.inc

  NWLAN_CONFIG = Record
    networdAddress : Array[0..3] Of nuint8;
    hostAddress : Array[0..5] Of nuint8;
    boardInstalled : nuint8;
    optionNumber : nuint8;
    configurationText1 : Array[0..79] Of nuint8;
    configurationText2 : Array[0..79] Of nuint8
  End;
```

Fields

networdAddress

hostAddress

boardInstalled

optionNumber

configurationText1

configurationText2

PACKET_BURST_INFO

Returns packet burst information

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   bigInvalidSlotCount;
   nuint32   bigForgedPacketCount;
   nuint32   bigInvalidPacketCount;
   nuint32   bigStillTransmittingCount;
   nuint32   stillDoingTheLastRequestCount;
   nuint32   invalidCtrlRequestCount;
   nuint32   ctrlInvalidMessageNumCount;
   nuint32   ctrlBeingTornDownCount;
   nuint32   bigRepeatTheFileReadCount;
   nuint32   bigSendExtraCCCount;
   nuint32   bigReturnAbortMessageCount;
   nuint32   bigReadInvalidMessageNumCount;
   nuint32   bigReadDoItOverCount;
   nuint32   bigReadBeingTornDownCount;
   nuint32   previousCtrlPacketCount;
   nuint32   sendHoldOffMessageCount;
   nuint32   bigReadNoDataAvailableCount;
   nuint32   bigReadTryingToReadTooMuchCount;
   nuint32   asyncReadErrorCount;
   nuint32   bigReadPhysicalReadErrorCount;
   nuint32   ctrlBadACKFragmentListCount;
   nuint32   ctrlNoDataReadCount;
   nuint32   writeDuplicateRequestCount;
   nuint32   shouldntBeACKingHereCount;
   nuint32   writeInconsistentPktLengthsCnt;
   nuint32   firstPacketIsntAWriteCount;
   nuint32   writeTrashedDuplicateRequestCnt;
   nuint32   bigWriteInvalidMessageNumCount;
   nuint32   bigWriteBeingTornDownCount;
   nuint32   bigWriteBeingAbortedCount;
   nuint32   zeroACKFragmentCountCount;
   nuint32   writeCurrentlyTransmittingCount;
   nuint32   tryingToWriteTooMuchCount;
   nuint32   writeOutOfMemForCtrlNodesCount;
   nuint32   writeDidntNeedThisFragmentCount;
   nuint32   writeTooManyBuffsCheckedOutCnt;
   nuint32   writeTimeOutCount;
   nuint32   writeGotAnACKCount;
   nuint32   writeGotAnACKCount1;
   nuint32   pollerAbortedTheConnCount;
   nuint32   maybeHadOutOfOrderWritesCount;
   nuint32   hadAnOutOfOrderWriteCount;
   nuint32   movedTheACKBitDownCount;
   nuint32   bumpedOutOfOrderWriteCount;
   nuint32   pollerRemovedOldOutOfOrderCount;
   nuint32   writeDidntNeedButRequestACKCnt;
   nuint32   writeTrashedPacketCount;
   nuint32   tooManyACKFragmentsCount;
   nuint32   savedAnOutOfOrderPacketCount;
   nuint32   connBeingAbortedCount;
} PACKET_BURST_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  PACKET_BURST_INFO = Record
    bigInvalidSlotCount : nuint32;
    bigForgedPacketCount : nuint32;
    bigInvalidPacketCount : nuint32;
    bigStillTransmittingCount : nuint32;
    stillDoingTheLastRequestCount : nuint32;
    invalidCtrlRequestCount : nuint32;
    ctrlInvalidMessageNumCount : nuint32;
    ctrlBeingTornDownCount : nuint32;
    bigRepeatTheFileReadCount : nuint32;
    bigSendExtraCCCount : nuint32;
    bigReturnAbortMessageCount : nuint32;
    bigReadInvalidMessageNumCount : nuint32;
    bigReadDoItOverCount : nuint32;
    bigReadBeingTornDownCount : nuint32;
    previousCtrlPacketCount : nuint32;
    sendHoldOffMessageCount : nuint32;
    bigReadNoDataAvailableCount : nuint32;
    bigReadTryingToReadTooMuchCount : nuint32;
    asyncReadErrorCount : nuint32;
    bigReadPhysicalReadErrorCount : nuint32;
    ctrlBadACKFragmentListCount : nuint32;
    ctrlNoDataReadCount : nuint32;
    writeDuplicateRequestCount : nuint32;
    shouldntBeACKingHereCount : nuint32;
    writeInconsistentPktLengthsCnt : nuint32;
    firstPacketIsntAWriteCount : nuint32;
    writeTrashedDuplicateRequestCnt : nuint32;
    bigWriteInvalidMessageNumCount : nuint32;
    bigWriteBeingTornDownCount : nuint32;
    bigWriteBeingAbortedCount : nuint32;
    zeroACKFragmentCountCount : nuint32;
    writeCurrentlyTransmittingCount : nuint32;
    tryingToWriteTooMuchCount : nuint32;
    writeOutOfMemForCtrlNodesCount : nuint32;
    writeDidntNeedThisFragmentCount : nuint32;
    writeTooManyBuffsCheckedOutCnt : nuint32;
    writeTimeOutCount : nuint32;
    writeGotAnACKCount : nuint32;
    writeGotAnACKCount1 : nuint32;
    pollerAbortedTheConnCount : nuint32;
    maybeHadOutOfOrderWritesCount : nuint32;
    hadAnOutOfOrderWriteCount : nuint32;
    movedTheACKBitDownCount : nuint32;
    bumpedOutOfOrderWriteCount : nuint32;
    pollerRemovedOldOutOfOrderCount : nuint32;
    writeDidntNeedButRequestACKCnt : nuint32;
    writeTrashedPacketCount : nuint32;
    tooManyACKFragmentsCount : nuint32;
    savedAnOutOfOrderPacketCount : nuint32;
    connBeingAbortedCount : nuint32
  End;
```

Fields

bigInvalidSlotCount

Specifies the number of requests determined not to be packet burst requests. The requests are either not NCP connections, the connection number is greater than available connection slots, or they have an invalid connection structure.

bigForgedPacketCount

Specifies the number of times the station connection ID and unique IDs do not match the server'.

bigInvalidPacketCount

Specifies the number of times the request packet is determined to be invalid.

bigStillTransmittingCount

Specifies the number of times the previous request is still transmitting so the current requests are delayed.

stillDoingTheLastRequestCount

Specifies the number of times the previous request is still being processed and read.

invalidCtrlRequestCount

Specifies the number of times the type of the packet and the packet nature cannot be determined.

ctrlInvalidMessageNumCount

Specifies the number of packets received that do not have a message number corresponding to what is already been sent and what is being inquired about.

ctrlBeingTornDownCount

Specifies the number of times the connection is torn down.

bigRepeatTheFileReadCount

Specifies the number of times an old request had to be reread.

bigSendExtraCCCount

Specifies the number of times the completion code error needed to be resent.

bigReturnAbortMessageCount

Specifies the number of times an 'Abort Message' was returned to the server.

bigReadInvalidMessageNumCount

Specifies the number of times the message number comparison between the station and server failed.

bigReadDoItOverCount

Specifies the number of times a request that had been processed was received again and reprocessed.

bigReadBeingTornDownCount

Specifies the number of times the status flag is set to abort so the read is torn down before being processed.

previousCtrlPacketCount

Specifies the number of times the server packet ID does not compare to the station's packet ID.

sendHoldOffMessageCount

Specifies the number of times the station's request

bigReadNoDataAvailableCount

Specifies the number of times nothing was able to be read from the request packet.

bigReadTryingToReadTooMuchCount

Specifies the number of times the request packet was greater than 64K so was unreadable.

asyncReadErrorCount

Specifies the number of times an error is returned trying to read the data in from the request packet.

bigReadPhysicalReadErrorCount

Specifies the number of times a physical read error was encountered while reading the request packet.

ctrlBadACKFragmentListCount

Specifies the number of times the fragments making up the packet burst exceeded the maximum fragments allowed.

ctrlNoDataReadCount

Specifies the number of times a control packet was received specifying to send more data after the request was thought to be over. The station error confirmed the error that no control data was read.

writeDuplicateRequestCount

Specifies the number of times a duplicate write was performed after a duplicate request was processed.

shouldntBeACKingHereCount

Specifies the number of times a request was being initialized when a fragment count was found indicating the station still needs to send some fragments.

writeInconsistentPktLengthsCnt

Specifies the number of times consistency checking on packet lengths failed.

firstPacketIsntAWriteCount

Specifies the number of times the first fragment packet is not a write request.

writeTrashedDuplicateRequestCnt

Specifies the number of times a duplicate write request was received without a need to acknowledge receipt of the request so it was trashed.

bigWriteInvalidMessageNumCount

Specifies the number of times the message number comparison between the station and the server failed.

bigWriteBeingTornDownCount

Specifies the number of times the write is aborted and the connection is torn down.

bigWriteBeingAbortedCount

Specifies the number of times the write is aborted and the station is cleared up.

zeroACKFragmentCountCount

Specifies the number of times the request needs to be retried so the acknowledge fragment count is zeroed out.

writeCurrentlyTransmittingCount

Specifies the number of times data was currently being transmitted so the transmission needed to complete before writing.

tryingToWriteTooMuchCount

Specifies the number of times an attempt was made to write a total length greater than 64K.

writeOutOfMemForCtrlNodesCount

Is currently unused.

writeDidntNeedThisFragmentCount

Specifies the number of times that everything needed is already acquired so the write process is ended.

writeTooManyBuffsCheckedOutCnt

Specifies the number of times the next packet was received without received the first packet. If the buffer cannot be kept, buffers are not released.

writeTimeOutCount

Specifies the number of times the write retry limit was exceeded.

writeGotAnACKCount

Specifies the number of times the client is resending part or all of the request back because the request timed out and there are missing pieces.

writeGotAnACKCount1

Specifies the number of times the client is resending part or all of the request back because the request timed out and there are missing pieces.

pollerAbortedTheConnCount

Specifies the number of times the poller lost track of the connection and the connection was cleared.

maybeHadOutOfOrderWritesCount

Specifies the number of times that extra packets for the write came in and needed to be reordered before a write completed.

hadAnOutOfOrderWriteCount

Specifies the number of times out of order packets were received and needed to be requested in the right order.

movedTheACKBitDownCount

Specifies the number of times out of order packets were reordered and information that was buffered up will not be re-requested by the out of order writes.

bumpedOutOfOrderWriteCount

Specifies the number of times packets needing to be reordered could not be reordered. The packets were thrown out and need to be resent.

pollerRemovedOldOutOfOrderCount

Specifies the number of times the write received buffers that were removed because they were on the out of order list too long.

writeDidntNeedButRequestACKCnt

Specifies the number of times the fragment count was zero indicating no new fragments were expected but the acknowledge bit was checked.

writeTrashedPacketCount

Specifies the number of times the cleaning up of additional receive buffers queued up during the execution of an error path.

tooManyACKFragmentsCount

Specifies the number of times a fragment placement needed to be adjusted.

savedAnOutOfOrderPacketCount

Specifies the number of times an out of order packet was received, saved for several seconds, and the buffer then moved onto the out of order write list.

connBeingAbortedCount

Specifies the number of times a station's connection was aborted.

PHYS_DSK_STATS

Returns physical disk statistics

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   systemElapsedTime;
   nuint8    diskChannel;
   nuint8    diskRemovable;
   nuint8    driveType;
   nuint8    controllerDriveNumber;
   nuint8    controllerNumber;
   nuint8    controllerType;
   nuint32   driveSize;/*in 4096 byte blocks*/
   nuint16   driveCylinders;
   nuint8    driveHeads;
   nuint8    sectorsPerTrack;
   nuint8    driveDefinition[64];
   nuint16   IOErrorCount;
   nuint32   hotFixStart;/*only with SFT I or greater*/
   nuint16   hotFixSize;/*only with SFT I or greater*/
   nuint16   hotFixBlockAvailable;/*only with SFT I or greater*/
   nuint8    hotFixDisabled;/*only with SFT I or greater*/
} PHYS_DSK_STATS;
```

Pascal Structure

```
Defined in nwserver.inc

PHYS_DSK_STATS = Record
    systemElapsedTime : nuint32;
    diskChannel : nuint8;
    diskRemovable : nuint8;
    driveType : nuint8;
    controllerDriveNumber : nuint8;
    controllerNumber : nuint8;
    controllerType : nuint8;
    driveSize : nuint32;
    driveCylinders : nuint16;
    driveHeads : nuint8;
    sectorsPerTrack : nuint8;
    driveDefinition : Array[0..63] Of nuint8;
    IOErrorCount : nuint16;
    hotFixStart : nuint32;
    hotFixSize : nuint16;
    hotFixBlockAvailable : nuint16;
    hotFixDisabled : nuint8
  End;
```

Fields

systemElapsedTime

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

diskChannel

Specifies the disk channel to which the disk unit is attached.

diskRemovable

Specifies whether a disk is removable (0 = nonremovable).

driveType

Specifies the type of drive, defined as follows:

```
1   XT
2   AT
3   SCSI
4   disk coprocessor
5   PS/2 with MFM Controller
6   PS/2 with ESDI Controller
7   Convergent Technology                SBIC
50.to.255   Value-Added Disk Drive
```

controllerDriveNumber

Specifies the drive number of the disk unit relative to the controller number.

controllerNumber

Specifies the address on the physical disk channel of the disk controller.

controllerType

Specifies the number identifying the type (make and model) of the disk controller.

driveSize

Specifies the size of the physical drive in blocks (1 block = 4,096 bytes). The drive size does not include the portion of the disk reserved for Hot Fix redirection in the event of media errors.

driveCylinders

Specifies the number of physical cylinders on the drive.

driveHeads

Specifies the number of disk heads on the drive.

sectorsPerTrack

Specifies the number of sectors on each disk track (1 sector = 512 bytes).

driveDefinition

Specifies the make and model of the drive (NULL-terminated string).

IOErrorCount

Specifies the number of times I/O errors have occurred on the disk since the server was brought up.

hotFixStart

Specifies the first block of the disk Hot Fix Redirection Table. This field is meaningful only with SFT(tm) NetWare Level I or above. The redirection table is used to replace bad disk blocks with usable blocks in the event that a media failure occurs on the disk

hotFixSize

Specifies the total number of redirection blocks set aside on the disk for Hot Fix redirection. Some or all of these blocks may be in use. *hotFixSize* is meaningful only with SFT NetWare Level I or above. To determine the number of redirection blocks still available for future use, see *hotFixBlocksAvailable*

hotFixBlockAvailable

Specifies the number of redirection blocks that are still available. *hotFixBlockAvailable* is meaningful only on SFT NetWare Level I or above.

hotFixDisabled

Specifies whether Hot Fix is enabled or disabled. *hotFixDisabled* is meaningful only with SFT NetWare Level I or above (0 = enabled).

resourceTagBuf

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   number;
   nuint32   signature;
   nuint32   count;
   nuint8    name);
```

Fields

number

signature

count

name

ROUTERS_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint8    nodeAddress[6];
   nuint32   connectedLAN;
   nuint16   routeHops;
   nuint16   routeTime;
} ROUTERS_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  ROUTERS_INFO = Record
    nodeAddress : Array[0..5] Of nuint8;
    connectedLAN : nuint32;
    routeHops : nuint16;
    routeTime : nuint16
  End;
```

Fields

nodeAddress

connectedLAN

routeHops

routeTime

SERVER_AND_VCONSOLE_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   currentServerTime;
   nuint8    vconsoleVersion;
   nuint8    vconsoleRevision;
} SERVER_AND_VCONSOLE_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

SERVER_AND_VCONSOLE_INFO = Record
   currentServerTime : nuint32;
   vconsoleVersion : nuint8;
   vconsoleRevision : nuint8
End;
```

Fields

currentServerTime

Specifies the time in ticks (about 1/18 second) since the server was brought up. When *currentServerTime* reaches 0xFFFFFFFF, it wraps to 0.

vconsoleVersion

Specifies the console version number and tracks the packet format.

vconsoleRevision

Specifies the console version revision number and tracks the packet format.

SERVER_LAN_IO_STATS

Returns LAN IO statistics for a LAN card on a NetWare server

Service: Server Environment

Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   systemElapsedTime;
   nuint16   maxRoutingBuffersAvail;
   nuint16   maxRoutingBuffersUsed;
   nuint16   routingBuffersInUse;
   nuint32   totalFileServicePackets;
   nuint16   fileServicePacketsBuffered;
   nuint16   invalidConnPacketCount;
   nuint16   badLogicalConnCount;
   nuint16   packetsRcvdDuringProcCount;
   nuint16   reprocessedRequestCount;
   nuint16   badSequenceNumberPacketCount;
   nuint16   duplicateReplyCount;
   nuint16   acknowledgementsSent;
   nuint16   badRequestTypeCount;
   nuint16   attachDuringProcCount;
   nuint16   attachWhileAttachingCount;
   nuint16   forgedDetachRequestCount;
   nuint16   badConnNumberOnDetachCount;
   nuint16   detachDuringProcCount;
   nuint16   repliesCanceledCount;
   nuint16   hopCountDiscardCount;
   nuint16   unknownNetDiscardCount;
   nuint16   noDGroupBufferDiscardCount;
   nuint16   outPacketNoBufferDiscardCount;
   nuint16   IPXNotMyNetworkCount;
   nuint32   NetBIOSPropagationCount;
   nuint32   totalOtherPackets;
   nuint32   totalRoutedPackets;
} SERVER_LAN_IO_STATS;
```

Pascal Structure

```
Defined in nwserver.inc

  SERVER_LAN_IO_STATS = Record
    systemElapsedTime : nuint32;
    maxRoutingBuffersAvail : nuint16;
    maxRoutingBuffersUsed : nuint16;
    routingBuffersInUse : nuint16;
```

```
      totalFileServicePackets : nuint32;
      fileServicePacketsBuffered : nuint16;
      invalidConnPacketCount : nuint16;
      badLogicalConnCount : nuint16;
      packetsRcvdDuringProcCount : nuint16;
      reprocessedRequestCount : nuint16;
      badSequenceNumberPacketCount : nuint16;
      duplicateReplyCount : nuint16;
      acknowledgementsSent : nuint16;
      badRequestTypeCount : nuint16;
      attachDuringProcCount : nuint16;
      attachWhileAttachingCount : nuint16;
      forgedDetachRequestCount : nuint16;
      badConnNumberOnDetachCount : nuint16;
      detachDuringProcCount : nuint16;
      repliesCanceledCount : nuint16;
      hopCountDiscardCount : nuint16;
      unknownNetDiscardCount : nuint16;
      noDGroupBufferDiscardCount : nuint16;
      outPacketNoBufferDiscardCount : nuint16;
      IPXNotMyNetworkCount : nuint16;
      NetBIOSPropagationCount : nuint32;
      totalOtherPackets : nuint32;
      totalRoutedPackets : nuint32
   End;
```

Fields

systemElapsedTime

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

maxRoutingBuffersAvail

Specifies the number of routing buffers the network is configured to handle.

maxRoutingBuffersUsed

Specifies the maximum number of routing buffers that have been in use simultaneously since the server was brought up.

routingBuffersInUse

Specifies the number of routing buffers that are being used by the server.

totalFileServicePackets

Specifies the number of request packets serviced by the NetWare server.

fileServicePacketsBuffered

Specifies the number of times file service request packets were stored in routing buffers.

invalidConnPacketCount

badLogicalConnCount

packetsRcvdDuringProcCount

Specifies the number of times a new request is received while the previous request is still being processed. Such packets are received when the client generates a duplicate request while the response to the first request is being sent to the client. In this case, the NetWare server will reprocess the request unnecessarily.

reprocessedRequestCount

Specifies a count of requests reprocessed by the server. Requests can be reprocessed if the client repeats a request for one that did not receive a response.

badSequenceNumberPacketCount

Specifies a count of request packets the server received from a connection whose packet sequence number does not match the current sequence number in the next sequence number. (Packets with bad sequence numbers are discarded.)

duplicateReplyCount

Specifies a count of request packets for which the server had to send a duplicate reply. (Duplicate replies are sent only for requests the server cannot process.)

acknowledgementsSent

Specifies the number of times a client repeats a request that is being serviced.

badRequestTypeCount

Specifies a count of request packets containing an invalid request type.

attachDuringProcCount

Specifies the number of times the server is requested to create a service connection by clients for which the server is currently processing a request. In this case, the server does not respond to the request currently being processed, and the server recreates a connection with the client (station).

attachWhileAttachingCount

Specifies the number of times the NetWare server receives a request to create a service connection while still servicing the same request received previously. Such duplicate requests are ignored.

forgedDetachRequestCount

Specifies the count of requests to terminate a connection whose source address does not match the address the server has assigned to the connection. The detach request is ignored.

badConnNumberOnDetachCount

Specifies the count of requests to terminate a connection for a connection number that is not supported by the server.

detachDuringProcCount

Specifies the number of requests to terminate a connection while requests are still being processed for the client.

repliesCanceledCount

Specifies the number of replies that were cancelled because the connection was reallocated during processing.

hopCountDiscardCount

Specifies the number of packets discarded because they have passed through more than 16 bridges without reaching their destination. (The maximum number of bridges might depend on the particular implementation of NCP, but for NetWare 2.x compatibility the maximum number of bridges should be 16.)

unknownNetDiscardCount

Specifies the number of packets that were discarded because their destination network is unknown to the server.

noDGroupBufferDiscardCount

Specifies the number of incoming packets that were received in a routing buffer that needed to be transferred to a DGroup buffer so that the socket dispatcher could transfer the packet to the correct socket. If no buffers are available, the packet is lost.

outPacketNoBufferDiscardCount

Specifies the number of packets the server attempted to send which were lost because no routing buffers were available.

IPXNotMyNetworkCount

Specifies the count of packets received that were destined for the B or C side drivers.

NetBIOSPropagationCount

Specifies a count of NetBIOS packets circulated through this network.

totalOtherPackets

Specifies a count of all packets received that were not requests for file services.

totalRoutedPackets

Specifies a count of all packets routed by the server.

SERVERS_SRC_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint8    serverNode[6];
   nuint32   connectedLAN;
   nuint16   sourceHops;
} SERVERS_SRC_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  SERVERS_SRC_INFO = Record
    serverNode : Array[0.. 6 -1] Of nuint8;
    connectedLAN : nuint32;
    sourceHops : nuint16
  End;
```

Fields

serverNode

Specifies the node address of the server.

connectedLAN

Specifies the LAN board number of the server.

sourceHops

Specifies the number of hops to the server.

SPX_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint16   SPXMaxConnsCount;
   nuint16   SPXMaxUsedConns;
   nuint16   SPXEstConnReq;
   nuint16   SPXEstConnFail;
   nuint16   SPXListenConnectReq;
   nuint16   SPXListenConnectFail;
   nuint32   SPXSendCount;
   nuint32   SPXWindowChokeCount;
   nuint16   SPXBadSendCount;
   nuint16   SPXSendFailCount;
   nuint16   SPXAbortedConn;
   nuint32   SPXListenPacketCount;
   nuint16   SPXBadListenCount;
   nuint32   SPXIncomingPacketCount;
   nuint16   SPXBadInPacketCount;
   nuint16   SPXSuppressedPackCount;
   nuint16   SPXNoSesListenECBCount;
   nuint16   SPXWatchDogDestSesCount;
} SPX_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 SPX_INFO = Record
    SPXMaxConnsCount : nuint16;
    SPXMaxUsedConns : nuint16;
    SPXEstConnReq : nuint16;
    SPXEstConnFail : nuint16;
    SPXListenConnectReq : nuint16;
    SPXListenConnectFail : nuint16;
    SPXSendCount : nuint32;
    SPXWindowChokeCount : nuint32;
    SPXBadSendCount : nuint16;
    SPXSendFailCount : nuint16;
    SPXAbortedConn : nuint16;
    SPXListenPacketCount : nuint32;
    SPXBadListenCount : nuint16;
    SPXIncomingPacketCount : nuint32;
    SPXBadInPacketCount : nuint16;
    SPXSuppressedPackCount : nuint16;
    SPXNoSesListenECBCount : nuint16;
    SPXWatchDogDestSesCount : nuint16
  End;
```

Fields

SPXMaxConnsCount

Specifies the maximum number of SPX™ connections allowed on the server.

SPXMaxUsedConns

Specifies the maximum number of SPX connections used at one time since the server was booted.

SPXEstConnReq

Specifies the total number of SPX connections established since the server was booted.

SPXEstConnFail

Specifies the number of times that an attempt to establish an SPX connection failed since the server was booted.

SPXListenConnectReq

Specifies the number of requests to post a listen since the server was booted.

SPXListenConnectFail

Specifies the number of times a request to post a listen failed since the server was booted.

SPXSendCount

Specifies the number of SPX packets sent since the server was booted.

SPXWindowChokeCount

Specifies the value used internally for debugging.

SPXBadSendCount

Specifies the number of bad packets sent since the server was booted.

SPXSendFailCount

Specifies the number of packets sent for which no acknowledgment was received since the server was booted.

SPXAbortedConn

Specifies the number of times a connection was aborted since the server was booted.

SPXListenPacketCount

Specifies the number of times a listen was posted on a socket since the server was booted.

SPXBadListenCount

Specifies the number of times a listen on a socket failed since the server was booted.

SPXIncomingPacketCount

Specifies the number of packets in the queue.

SPXBadInPacketCount

Specifies the number of bad SPX packets received since the server was booted.

SPXSuppressedPackCount

Specifies the number of times a duplicate SPX packet was received since the server was booted.

SPXNoSesListenECBCount

Specifies the number of times a listen was posted on a session that was not established since the server was booted.

SPXWatchDogDestSesCount

Specifies the number of times the watchdog destroyed a session since the server was booted.

STACK_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   StackNum;
   nuint8    StackShortName[16];
} STACK_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

STACK_INFO = Record
    StackNum : nuint32;
    StackShortName : Array[0..15] Of nuint8
  End;
```

Fields

StackNum

Specifies the protocol number.

StackShortName

Specifies the protocol short name with *StackShortName* being the length, and the rest of the name following. It is not null-terminated.

USER_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   connNum;
   nuint32   useCount;
   nuint8    connServiceType;
   nuint8    loginTime[7];
   nuint32   status;
   nuint32   expirationTime;
   nuint32   objType;
   nuint8    transactionFlag;
   nuint8    logicalLockThreshold;
   nuint8    recordLockThreshold;
   nuint8    fileWriteFlags;
   nuint8    fileWriteState;
   nuint8    filler;
   nuint16   fileLockCount;
   nuint16   recordLockCount;
   nuint8    totalBytesRead[6];
   nuint8    totalBytesWritten[6];
   nuint32   totalRequests;
   nuint32   heldRequests;
   nuint8    heldBytesRead[6];
   nuint8    heldBytesWritten[6];
} USER_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

 USER_INFO = Record
    connNum : nuint32;
    useCount : nuint32;
    connServiceType : nuint8;
    loginTime : Array[0..6] Of nuint8;
    status : nuint32;
    expirationTime : nuint32;
    objType : nuint32;
    transactionFlag : nuint8;
    logicalLockThreshold : nuint8;
    recordLockThreshold : nuint8;
    fileWriteFlags : nuint8;            (* Includes active and stop bit
    fileWriteState : nuint8;
    filler : nuint8;
    fileLockCount : nuint16;
    recordLockCount : nuint16;
    totalBytesRead : Array[0..5] Of nuint8;
    totalBytesWritten : Array[0..5] Of nuint8;
    totalRequests : nuint32;
    heldRequests : nuint32;
    heldBytesRead : Array[0..5] Of nuint8;
    heldBytesWritten : Array[0..5] Of nuint8;
  End;
```

Fields

connNum

Specifies the connection number of the user.

useCount

Specifies if the connection is in use: 1 if the connection is in use, 0 if the connection is not in use.

connServiceType

Specifies the following connection types:

```
2 = FSE_NCP_CONNECTION_TYPE
3 = FSE_NLM_CONNECTION_TYPE
4 = FSE_AFP_CONNECTION_TYPE
5 = FSE_FTAM_CONNECTION_TYPE
6 = FSE_ANCP_CONNECTION_TYPE
```

loginTime

Specifies the time the user logged in.

status

Specifies the status of the connection:

```
LOGGED_IN                     0x00000001
BEING_ABORTED                 0x00000002
AUDITED                       0x00000004
NEEDS_SECURITY_CHANGE         0x00000008
MAC_STATION                   0x00000010
AUTHENTICATED_TEMPORARY       0x00000020
AUDIT_CONNECTION_RECORDED     0x00000040
DSAUDIT_CONNECTION_RECORDED   0x00000080
```

expirationTime

objType

Specifies the object type of the user, usually 0x0100.

transactionFlag

Specifies the transaction tracking information.

logicalLockThreshold

Specifies the maximum number of logical locks a user an have.

recordLockThreshold

Specifies the maximum number of record locks the user can have.

fileWriteFlags

Specifies the writing status (includes active and stop bits) as follows:

```
1 = WRITING
2 = WRITING_ABORTED
```

fileWriteState

Specifies the writing status as follows:

```
0 = NOT_WRITING
1 = WRITE_IN_PROGRESS
2 = WRITE_BEING_STOPPED
```

filler

Is unused.

fileLockCount

Specifies the number of files the user has locked.

recordLockCount

Specifies the number of records the user has locked.

totalBytesRead

Specifies the number of bytes the user has read (48-bit value).

totalBytesWritten

Specifies the number of bytes the user has written (48-bit value).

totalRequests

Specifies the number of requests the user has sent.

heldRequests

Specifies the number of requests held for accounting purposes.

heldBytesRead

Specifies the number of bytes the user has read that have a hold on them for accounting purposes.

heldBytesWritten

Specifies the number of bytes the user has written that have a hold on them for accounting purposes

VERSION_INFO

Returns version information for a NetWare server's logical components

Service: Server Environment

Defined In: nwfserver.h

Structure

```
typedef struct
{
   nuint8    serverName[48];
   nuint8    fileServiceVersion;
   nuint8    fileServiceSubVersion;
   nuint16   maximumServiceConnections;
   nuint16   connectionsInUse;
   nuint16   maxNumberVolumes;
   nuint8    revision;
   nuint8    SFTLevel;
   nuint8    TTSLevel;
   nuint16   maxConnectionsEverUsed;
   nuint8    accountVersion;
   nuint8    VAPVersion;
   nuint8    queueVersion;
   nuint8    printVersion;
   nuint8    virtualConsoleVersion;
   nuint8    restrictionLevel;
   nuint8    internetBridge;
   nuint8    reserved[60];
} VERSION_INFO;
```

Pascal Structure

```
Defined in nwserver.inc

VERSION_INFO = Record
    serverName : Array[0..47] Of nuint8;
    fileServiceVersion : nuint8;
    fileServiceSubVersion : nuint8;
    maximumServiceConnections : nuint16;
    connectionsInUse : nuint16;
    maxNumberVolumes : nuint16;
    revision : nuint8;
    SFTLevel : nuint8;
    TTSLevel : nuint8;
    maxConnectionsEverUsed : nuint16;
    accountVersion : nuint8;
    VAPVersion : nuint8;
    queueVersion : nuint8;
    printVersion : nuint8;
    virtualConsoleVersion : nuint8;
    restrictionLevel : nuint8;
    internetBridge : nuint8;
    reserved : Array[0..59] Of nuint8
  End;
```

Fields

serverName

fileServiceVersion

fileServiceSubVersion

maximumServiceConnections

connectionsInUse

maxNumberVolumes

revision

SFTLevel

TTSLevel

maxConnectionsEverUsed

accountVersion

VAPVersion

queueVersion

printVersion

virtualConsoleVersion

restrictionLevel

internetBridge

reserved

VOLUME_INFO_BY_LEVEL

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef union
{
   VOLUME_INFO_BY_LEVEL_DEF     volInfoDef;
   VOLUME_INFO_BY_LEVEL_DEF2    volInfoDef2;
} VOLUME_INFO_BY_LEVEL;
```

Fields

volInfoDef

volInfoDef2

VOLUME_INFO_BY_LEVEL_DEF

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   volumeType;
   nuint32   statusFlagBits;
   nuint32   sectorSize;
   nuint32   sectorsPerCluster;
   nuint32   volumeSizeInClusters;
   nuint32   freedClusters;
   nuint32   subAllocFreeableClusters;
   nuint32   freeableLimboSectors;
   nuint32   nonFreeableLimboSectors;
   nuint32   nonFreeableAvailSubAllocSectors;
   nuint32   notUsableSubAllocSectors;
   nuint32   subAllocClusters;
   nuint32   dataStreamsCount;
   nuint32   limboDataStreamsCount;
   nuint32   oldestDeletedFileAgeInTicks;
   nuint32   compressedDataStreamsCount;
   nuint32   compressedLimboDataStreamsCount;
   nuint32   unCompressableDataStreamsCount;
   nuint32   preCompressedSectors;
   nuint32   compressedSectors;
   nuint32   migratedFiles;
   nuint32   migratedSectors;
   nuint32   clustersUsedByFAT;
   nuint32   clustersUsedByDirectories;
   nuint32   clustersUsedbyExtendedDirs;
   nuint32   totalDirectoryEntries;
   nuint32   unUsedDirectoryEntries;
   nuint32   totalExtendedDirectoryExtants;
   nuint32   unUsedExtendedDirectoryExtants;
   nuint32   extendedAttributesDefined;
   nuint32   extendedAttributeExtantsUsed;
   nuint32   directoryServicesObjectID;
   nuint32   volumeLastModifiedDateAndTime;
} VOLUME_INFO_BY_LEVEL_DEF;
```

Pascal Structure

```
Defined in nwfse.inc

  VOLUME_INFO_BY_LEVEL_DEF = Record
```

```
      volumeType : nuint32;
      statusFlagBits : nuint32;
      sectorSize : nuint32;
      sectorsPerCluster : nuint32;
      volumeSizeInClusters : nuint32;
      freedClusters : nuint32;
      subAllocFreeableClusters : nuint32;
      freeableLimboSectors : nuint32;
      nonFreeableLimboSectors : nuint32;
      nonFreeableAvailSubAllocSectors : nuint32;
      notUsableSubAllocSectors : nuint32;
      subAllocClusters : nuint32;
      dataStreamsCount : nuint32;
      limboDataStreamsCount : nuint32;
      oldestDeletedFileAgeInTicks : nuint32;
      compressedDataStreamsCount : nuint32;
      compressedLimboDataStreamsCount : nuint32;
      unCompressableDataStreamsCount : nuint32;
      preCompressedSectors : nuint32;
      compressedSectors : nuint32;
      migratedFiles : nuint32;
      migratedSectors : nuint2;
      clustersUsedByFAT : nunt32;
      clustersUsedByDirectories : nuint32;
      clustersUsedByExtendedDirs : nuint32;
      totalDirectoryEntries : nuint32;
      unUsedDirectoryEntries : nuint32;
      totalExtendedDirectoryExtants : nuint32;
      unUsedExtendedDirectoryExtants : nuint32;
      extendedAttributesDefined : nuint32;
      extendedAttributeExtantsUsed : nuint32;
      directoryServicesObjectID : nuint32;
      volumeLastModifiedDateAndTime : nuint32
   End;
```

Fields

volumeType

statusFlagBits

sectorSize

sectorsPerCluster

volumeSizeInClusters

freedClusters

subAllocFreeableClusters

freeableLimboSectors

nonFreeableLimboSectors

nonFreeableAvailSubAllocSectors

notUsableSubAllocSectors

subAllocClusters

dataStreamsCount

limboDataStreamsCount

oldestDeletedFileAgeInTicks

compressedDataStreamsCount

compressedLimboDataStreamsCount

unCompressableDataStreamsCount

preCompressedSectors

compressedSectors

migratedFiles

migratedSectors

clustersUsedByFAT

clustersUsedByDirectories

clustersUsedbyExtendedDirs

totalDirectoryEntries

unUsedDirectoryEntries

totalExtendedDirectoryExtants

unUsedExtendedDirectoryExtants

extendedAttributesDefined

extendedAttributeExtantsUsed

directoryServicesObjectID

volumeLastModifiedDateAndTime

VOLUME_INFO_BY_LEVEL_DEF2

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   volumeActiveCount;
   nuint32   volumeUseCount;
   nuint32   mACRootIDs;
   nuint32   volumeLastModifiedDateAndTime;
   nuint32   volumeReferenceCount;
   nuint32   compressionLowerLimit;
   nuint32   outstandingIOs;
   nuint32   outstandingCompressionIOs;
   nuint32   compressionIOsLimit;
} VOLUME_INFO_BY_LEVEL_DEF2;
```

Pascal Structure

```
Defined in nwfse.inc

 VOLUME_INFO_BY_LEVEL_DEF2 = Record
    volumeActiveCount : nuint32;
    volumeUseCount : nuint32;
    mACRootIDs : nuint32;
    volumeLastModifiedDateAndTime : nuint32;
    volumeReferenceCount : nuint32;
    compressionLowerLimit : nuint32;
    outstandingIOs : nuint32;
    outstandingCompressionIOs : nuint32;
    compressionIOsLimit : nuint32
  End;
```

Fields

volumeActiveCount

volumeUseCount

mACRootIDs

volumeLastModifiedDateAndTime

volumeReferenceCount

compressionLowerLimit

outstandingIOs

outstandingIOs

outstandingCompressionIOs

compressionIOsLimit

VOLUME_SEGMENT

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   volumeSegmentDeviceNum;
   nuint32   volumeSegmentOffset;
   nuint32   volumeSegmentSize;
} VOLUME_SEGMENT;
```

Pascal Structure

```
Defined in nwfse.inc

  VOLUME_SEGMENT = Record
    volumeSegmentDeviceNum : nuint32;
    volumeSegmentOffset : nuint32;
    volumeSegmentSize : nuint32
  End;
```

Fields

volumeSegmentDeviceNum

Specifies the device the segment is located on.

volumeSegmentOffset

Specifies the offset of the segment in bytes.

volumeSegmentSize

Specifies the segment size in bytes.

VOLUME_SWITCH_INFO

Service: Server Environment
Defined In: nwfse.h

Structure

```
typedef struct
{
   nuint32   readFile;
   nuint32   writeFile;
   nuint32   deleteFile;
   nuint32   renMove;
   nuint32   openFile;
   nuint32   createFile;
   nuint32   createAndOpenFile;
   nuint32   closeFile;
   nuint32   scanDeleteFile;
   nuint32   salvageFile;
   nuint32   purgeFile;
   nuint32   migrateFile;
   nuint32   deMigrateFile;
   nuint32   createDir;
   nuint32   deleteDir;
   nuint32   directoryScans;
   nuint32   mapPathToDirNum;
   nuint32   modifyDirEntry;
   nuint32   getAccessRights;
   nuint32   getAccessRightsFromIDs;
   nuint32   mapDirNumToPath;
   nuint32   getEntryFromPathStrBase;
   nuint32   getOtherNSEntry;
   nuint32   getExtDirInfo;
   nuint32   getParentDirNum;
   nuint32   addTrusteeR;
   nuint32   scanTrusteeR;
   nuint32   delTrusteeR;
   nuint32   purgeTrust;
   nuint32   findNextTrusteRef;
   nuint32   scanUserRestNodes;
   nuint32   addUserRest;
   nuint32   deleteUserRest;
   nuint32   rtnDirSpaceRest;
   nuint32   getActualAvailDskSp;
   nuint32   cntOwnedFilesAndDirs;
   nuint32   migFileInfo;
   nuint32   volMigInfo;
   nuint32   readMigFileData;
   nuint32   getVolUsageStats;
   nuint32   getActualVolUsageStats;
   nuint32   getDirUsageStats;
   nuint32   NMFileReadsCount;
   nuint32   NMFileWritesCount;
   nuint32   mapPathToDirNumOrPhantom;
   nuint32   stationHasAccessRgtsGntedBelow;
   nuint32   gtDataStreamLensFromPathStrBase;
   nuint32   checkAndGetDirectoryEntry;
   nuint32   getDeletedEntry;
   nuint32   getOriginalNameSpace;
   nuint32   getActualFileSize;
   nuint32   verifyNameSpaceNumber;
   nuint32   verifyDataStreamNumber;
   nuint32   checkVolumeNumber;
   nuint32   commitFile;
   nuint32   VMGetDirectoryEntry;
   nuint32   createDMFileEntry;
   nuint32   renameNameSpaceEntry;
   nuint32   logFile;
   nuint32   releaseFile;
   nuint32   clearFile;
   nuint32   setVolumeFlag;
   nuint32   clearVolumeFlag;
   nuint32   getOriginalInfo;
   nuint32   createMigratedDir;
   nuint32   F3OpenCreate;
   nuint32   F3InitFileSearch;
   nuint32   F3ContinueFileSearch;
   nuint32   F3RenameFile;
   nuint32   F3ScanForTrustees;
   nuint32   F3ObtainFileInfo;
   nuint32   F3ModifyInfo;
   nuint32   F3EraseFile;
   nuint32   F3SetDirHandle;
   nuint32   F3AddTrustees;
   nuint32   F3DeleteTrustees;
   nuint32   F3AllocDirHandle;
   nuint32   F3ScanSalvagedFiles;
   nuint32   F3RecoverSalvagedFiles;
   nuint32   F3PurgeSalvageableFile;
   nuint32   F3GetNSSpecificInfo;
   nuint32   F3ModifyNSSpecificInfo;
   nuint32   F3SearchSet;
   nuint32   F3GetDirBase;
   nuint32   F3QueryNameSpaceInfo;
   nuint32   F3GetNameSpaceList;
   nuint32   F3GetHugeInfo;
   nuint32   F3SetHugeInfo;
   nuint32   F3GetFullPathString;
   nuint32   F3GetEffectiveDirectoryRights;
} VOLUME_SWITCH_INFO;
```

Pascal Structure

```
Defined in nwfse.inc

  VOLUME_SWITCH_INFO = Record
    readFile : nuint32;
    writeFile : nuint32;
    deleteFile : nuint32;
    renMove : nuint32;
    openFile : nuint32;
    createFile : nuint32;
    createAndOpenFile : nuint32;
    closeFile : nuint32;
    scanDeleteFile : nuint32;
    salvageFile : nuint32;
    purgeFile : nuint32;
    migrateFile : nuint32;
    deMigrateFile : nuint32;
    createDir : nuint32;
    deleteDir : nuint32;
    directoryScans : nuint32;
    mapPathToDirNum : nuint32;
    modifyDirEntry : nuint32;
    getAccessRights : nuint32;
    getAccessRightsFromIDs : nuint32;
    mapDirNumToPath : nuint32;
    getEntryFromPathStrBase : nuint32;
    getOtherNSEntry : nuint32;
    getExtDirInfo : nuint32;
    getParentDirNum : nuint32;
    addTrusteeR : nuint32;
    scanTrusteeR : nuint32;
    delTrusteeR : nuint32;
    purgeTrust : nuint32;
    findNextTrustRef : nuint32;
    scanUserRestNodes : nuint32;
    addUserRest : nuint32;
    deleteUserRest : nuint2;
    rtnDirSpaceRest : nuint32;
    getActualAvailDskSp : nuint32;
    cntOwnedFilesAndDirs : nuint32;
    migFileInfo : nuint32;
    volMigInfo : nuint32;
    readMigFileData : nuint32;
    getVolUsageStats : nuint32;
    getActualVolUsageStats : nuint32;
    getDirUsageStats : nuint32;
    NMFileReadsCount : nuint32;
    NMFileWritesCount : nuint32;
    mapPathToDirNumOrPhantom : nuint32;
    stationHasAccessRgtsGntedBelow : nuint32;
    gtDataStreamLensFromPathStrBase : nuint32;
```

```
      checkAndGetDirectoryEntry : nuint32;
      getDeletedEntry : nuint32;
      getOriginalNameSpace : nuint32;
      getActualFileSize : nuint32;
      verifyNameSpaceNumber : nuint32;
      verifyDataStreamNumber : nuint32;
      checkVolumeNumber : nuint32;
      commitFile : nuint32;
      VMGetDirectoryEntry : nuint32;
      createDMFileEntry : nuint32;
      renameNameSpaceEntry : nuint32;
      logFile : nuint32;
      releaseFile : nuint32;
      clearFile : nuint32;
      setVolumeFlag : nuint32;
      clearVolumeFlag : nuint32;
      getOriginalInfo : nuint32;
      createMigratedDir : nuint32;
      F3OpenCreate : nuint32;
      F3InitFileSearch : nuint32;
      F3ContinueFileSearch : nuint32;
      F3RenameFile : nuint32;
      F3ScanForTrustees : nuint32;
      F3ObtainFileInfo : nuint32;
      F3ModifyInfo : nuint32;
      F3EraseFile : nuint32;
      F3SetDirHandle : nuint32;
      F3AddTrustees : nuint32;
      F3DeleteTrustees : nuint32;
      F3AllocDirHandle : nuint32;
      F3ScanSalvagedFiles : nuint32;
      F3RecoverSalvagedFiles : nuint32;
      F3PurgeSalvageableFile : nuint32;
      F3GetNSSpecificInfo : nuint32;
      F3ModifyNSSpecificInfo : nuint32;
      F3SearchSet : nuint32;
      F3GetDirBase : nuint32;
      F3QueryNameSpaceInfo : nuint32;
      F3GetNameSpaceList : nuint32;
      F3GetHugeInfo : nuint32;
      F3SetHugeInfo : nuint32;
      F3GetFullPathString : nuint32;
      F3GetEffectiveDirectoryRights : nuint32
    End;
```

Fields

readFile

writeFile

deleteFile

renMove

openFile

createFile

createAndOpenFile

closeFile

scanDeleteFile

salvageFile

purgeFile

migrateFile

deMigrateFile

createDir

deleteDir

directoryScans

mapPathToDirNum

modifyDirEntry

getAccessRights

getAccessRightsFromIDs

mapDirNumToPath

getEntryFromPathStrBase

getOtherNSEntry

getExtDirInfo

getParentDirNum

addTrusteeR

scanTrusteeR

delTrusteeR

purgeTrust

findNextTrusteRef

scanUserRestNodes

addUserRest

deleteUserRest

rtnDirSpaceRest

getActualAvailDskSp

cntOwnedFilesAndDirs

cntOwnedFilesAndDirs

volMigInfo

readMigFileData

getVolUsageStats

getActualVolUsageStats

getDirUsageStats

NMFileReadsCount

NMFileWritesCount

mapPathToDirNumOrPhantom

stationHasAccessRgtsGntedBelow

gtDataStreamLensFromPathStrBase

checkAndGetDirectoryEntry

getDeletedEntry

getOriginalNameSpace

getActualFileSize

verifyNameSpaceNumber

verifyDataStreamNumber

checkVolumeNumber

commitFile

VMGetDirectoryEntry

createDMFileEntry

renameNameSpaceEntry

logFile

releaseFile

clearFile

setVolumeFlag

clearVolumeFlag

getOriginalInfo

createMigratedDir

F3OpenCreate

F3InitFileSearch

F3ContinueFileSearch

F3RenameFile

F3ScanForTrustees

F3ObtainFileInfo

F3ModifyInfo

F3EraseFile

F3SetDirHandle

F3AddTrustees

F3DeleteTrustees

F3AllocDirHandle

F3ScanSalvagedFiles

F3RecoverSalvagedFiles

F3PurgeSalvageableFile

F3GetNSSpecificInfo

F3ModifyNSSpecificInfo

F3SearchSet

F3GetDirBase

F3QueryNameSpaceInfo

F3GetNameSpaceList

F3GetHugeInfo

F3SetHugeInfo

F3GetFullPathString

F3GetEffectiveDirectoryRights

Server Management

Server Management: Guides

Server Management: General Guide

Tasks

Managing Volumes

Managing a Volume's Name Space

Managing NCF Files

Managing NLMs

Managing SET Values

Concepts

Server Management Introduction

Related Topics:

Server Management: Tasks

Server Management: Concepts

Server Management: Functions

Parent Topic:

Management Overview

Server Management: Tasks

Managing a Volume’s Name Space

1. **To add a specified name space to a volume on a server, call NWSMAddNSToVolume.**

Parent Topic:

Server Management: General Guide

Managing NCF Files

1. **To execute a selected NCF file on a specified server, call NWSMExecuteNCFFile.**

Parent Topic:

Server Management: General Guide

Managing NLMs

1. **To load a selected NLM on a server, call NWSMLoadNLM.**
2. **To unload a selected NLM on a server, call NWSMUnloadNLM.**

Parent Topic:

Server Management: General Guide

Managing SET Values

1. **To change a SET integer value, call NWSMSetDynamicCmdIntValue.**
2. **To change a SET string value, call NWSMSetDynamicCmdStrValue.**

Parent Topic:

Server Management: General Guide

Managing Volumes

1. **To mount a volume using either a volume name or a volume number, call NWSMMountVolume.**

2. **To dismount a volume using a volume name, call NWSMDismountVolumeByName.**

3. **To dismount a volume using a volume number, call NWSMDismountVolumeByNumber.**

Parent Topic:

Server Management: General Guide

Server Management: Concepts

Server Management Introduction

Server Management allows you to manage server functions from the workstation.

From the workstation you can:

Manage a volume's name space
Manage volumes
Manage NCF files on a server
Manage NLMs
Manage SET command values.

Parent Topic:

Server Management: General Guide

Server Management: Functions

NWSMAddNSToVolume

Adds a specified name space to a mounted volume on a server
NetWare Server: 4.1
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT
Service: Server Management

Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMAddNSToVolume
  (NWCONN_HANDLE    connHandle,
   nuint16          volNumber,
   nuint8           namspc);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMAddNSToVolume
  (connHandle : NWCONN_HANDLE;
   volNumber : nuint16;
   namspc : nuint8
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

volNumber

(IN) Specifies the volume number on which the name space will be loaded.

namspc

(IN) Specifies the name space to load on the volume.

Return Values

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

0x0000	SUCCESS

0x00BF	No add name space string given
0x0205	Unable to add specified name space to a volume
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST
0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

You must be logged into *connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMAddNSToVolume**.

namspc values cannot be ORed; they may be added on each call. *namspc* values follow:

NW_NS_MAC
NW_NS_NFS
NW_NS_FTAM
NW_NS_OS2

Execute ADD NAME SPACE only once for each non-DOS naming convention you want to store on a volume.

Before you can add a name space to a volume, the volume and the name space module must both be loaded. See the **NWSMLoadNLM** function.

NCP Calls

0x2222 135 5 Add Name Space To Volume

See Also

NWSMLoadNLM

NWSMDismountVolumeByName

Dismounts a volume on a selected server
NetWare Server: 4.1
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT
Service: Server Management

Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMDismountVolumeByName
   (NWCONN_HANDLE     connHandle,
    pnstr8            volumeName);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMDismountVolumeByName
   (connHandle : NWCONN_HANDLE;
    volumeName : pnstr8
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

volumeName

(IN) Points to the name of the NetWare volume to dismount. It must be NULL terminated.

Return Values

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

0x0000	SUCCESS
0x00BF	Invalid *volumeName* string passed
0x0204	Unable to dismount specified volume name
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST

0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

You must be logged into*connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMDismountVolumeByName**.

NCP Calls

0x2222 131 4   Dismount Volume

See Also

NWSMMountVolume

NWSMDismountVolumeByNumber

Dismounts a volume on a selected server
NetWare Server: 4.1
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT
Service: Server Management

Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMDismountVolumeByNumber
   (NWCONN_HANDLE    connHandle,
    nuint16          volumeNumber);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMDismountVolumeByNumber
   (connHandle : NWCONN_HANDLE;
    volumeNumber : nuint16
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

volumeNumber

(IN) Specifies the number of the NetWare volume to dismount.

Return Values

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

0x0000	SUCCESS
0x00BF	Invalid *volumeName* string passed
0x0204	Unable to dismount specified volume name
0x8801	INVALID_CONNECTION
0x8998	VOLUME_DOES_NOT_EXIST

0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

You must be logged into*connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMDismountVolumeByNumber**.

NCP Calls

0x2222 131 4	Dismount Volume

See Also

NWSMMountVolume

NWSMExecuteNCFFile

Executes a selected NCF file on a specified server

NetWare Server: 4.1

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

Service: Server Management

Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMExecuteNCFFile
   (NWCONN_HANDLE   connHandle,
    pnstr8          NCFFileName);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMExecuteNCFFile
   (connHandle : NWCONN_HANDLE;
    NCFFileName : pnstr8
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

NCFFileName

(IN) Points to the name of the file to execute. It must be NULL terminated.

Return Values

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

0x0000	SUCCESS
0x009E	No Path and Name field supplied
0x0207	Unable to execute NCF file
0x8801	INVALID_CONNECTION
0x8993	INVALID_FILENAME

0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

You must be logged into *connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMExecuteNCFFile**.

fileName may include a volume and path in the following format:

```
{VOLUME NAME:}{PATH\...|file name}
```

NCP Calls

0x2222 131 7 Execute NCF File

NWSMLoadNLM

Loads a selected NLM on a server

NetWare Server: 4.1

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

Service: Server Management

Syntax

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

N_EXTERN_LIBRARY NWCCODE NWSMLoadNLM
   (NWCONN_HANDLE    connHandle,
    pnstr8           loadCommand);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMLoadNLM
   (connHandle : NWCONN_HANDLE;
    loadCommand : pnstr8
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

loadCommand

(IN) Points to the load command for the NLM. It must be NULL terminated; maximum length is 482 bytes.

Return Values

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

0x0000	SUCCESS
0x009E	Bad file name or no file name given
0x0202	Unable to load specified module
0x8801	INVALID_CONNECTION
0x88FB	ERR_NCP_NOT_SUPPORTED

0x899E	INVALID_FILENAME

Remarks

You must be logged into *connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMLoadNLM**.

loadCommand can be any string that is valid for the LOAD command on the console (except it does not include the actual “LOAD” command). It contains the NLM name (including the path if it is not in the server path), and command line parameters if any.

The LOAD command has the following format:

```
{VOLUME NAME:}{PATH\...}NLMname{.ext}{parameters}
```

NWSMLoadNLM does not support UNC paths.

The NLM does not have to include the extensions. .NLM will be assumed if the extensions are not included.

The server console LOAD command is documented in the NetWare server utilities documentation.

NCP Calls

0x2222 131 1   Load A NLM

See Also

NWSMUnloadNLM

NWSMMountVolume

Mounts a volume on a selected server

NetWare Server: 4.1

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

Service: Server Management

Syntax

```
#include <nwsm.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY NWCCODE NWSMMountVolume
  (NWCONN_HANDLE    connHandle,
   pnstr8           volumeName,
   pnuint32         volumeNumber);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMMountVolume
  (connHandle : NWCONN_HANDLE;
   volumeName : pnstr8;
   volumeNumber : pnuint32
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

volumeName

(IN) Points to the name of the NetWare volume to mount. It must be NULL terminated.

volumeNumber

(OUT) Points to the number of the volume.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESS

0x00BF	Invalid *volumeName* string passed
0x0203	Unable to mount specified volume
0x8801	INVALID_CONNECTION
0x89FB	ERR_NCP_NOT_SUPPORTED
0x8998	VOLUME_DOES_NOT_EXIST
	VOLUME_ALREADY_MOUNTED

Remarks

You must be logged into*connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMMountVolume**.

If upon mounting a volume, an error occurs, and if the set parameters are such that VREPAIR will automatically execute, VREPAIR will execute and **NWSMMountVolume** will not return until VREPAIR has completed. The volume is then mounted.

NCP Calls

0x2222 131 3   Mount Volume

See Also

NWSMDismountVolumeByName, **NWSMDismountVolumeByNumber**

NWSMSetDynamicCmdIntValue

Changes the current value of a set command (that takes in integer values) on a specified server

NetWare Server: 4.1

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT

Service: Server Management

Syntax

```
#include <nwsm.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY NWCCODE NWSMSetDynamicCmdIntValue
   (NWCONN_HANDLE    connHandle,
    pnstr8           setCommandName,
    nuint32          cmdValue);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMSetDynamicCmdIntValue
   (connHandle : NWCONN_HANDLE;
    setCommandName : pnstr8;
    cmdValue : nuint32
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

setCommandName

(IN) Points to the set parameter name. It must be NULL terminated.

cmdValue

(IN) Specifies the new value for the set command parameter.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESS

0x008C	Invalid type flag value
0x00BF	No *setCommandName* string
0x0206	Unable to set the command
0x8801	INVALID_CONNECTION
0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

You must be logged into *connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMSetDynamicCmdIntValue**.

The server console SET command is documented in the NetWare server utilities documentation. The SET values OFF/ON are treated as the integers 0 (zero) and 1 respectively; they are not treated as strings.

NCP Calls

0x2222 131 6 Set Command Value

See Also

NWGetServerSetCommandsInfo, **NWGetServerSetCategories**, **NWSMSetDynamicCmdStrValue**

NWSMSetDynamicCmdStrValue

Changes the current values of a set command (that takes in string values) on a specified server

NetWare Server: 4.1

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT

Service: Server Management

Syntax

```
#include <nwsm.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY NWCCODE NWSMSetDynamicCmdStrValue
  (NWCONN_HANDLE   connHandle,
   pnstr8          setCommandName,
   pnstr8          cmdValue);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMSetDynamicCmdStrValue
  (connHandle : NWCONN_HANDLE;
   setCommandName : pnstr8;
   cmdValue : pnstr8
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

setCommandName

(IN) Points to the parameter command name. It must be NULL terminated.

cmdValue

(IN) Points to the new value for the set command parameter. It must be NULL terminated.s

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESS
0x008C	Invalid type flag value
0x00BF	No *setCommandName* string
0x0206	Unable to set the command
0x8801	INVALID_CONNECTION
0x89FB	ERR_NCP_NOT_SUPPORTED

Remarks

You must be logged into *connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMSetDynamicCmdStrValue**.

The server console SET command is documented in the NetWare server utilities documentation. The SET values OFF/ON are treated as the integers 0 (zero) and 1 respectively; they are not treated as strings.

NCP Calls

0x2222 131 6 Set Command Value

See Also

NWGetServerSetCommandsInfo, **NWGetServerSetCategories**, **NWSMSetDynamicCmdIntValue**

NWSMUnloadNLM

Unloads a selected NLM on a server

NetWare Server: 4.1

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT

Service: Server Management

Syntax

```
#include <nwsm.h>
or
#include <nwcalls.h>

N_EXTERN_LIBRARY NWCCODE NWSMUnloadNLM
  (NWCONN_HANDLE    connHandle,
   pnstr8           NLMName);
```

Pascal Syntax

```
#include <nwsm.inc>

Function NWSMUnloadNLM
  (connHandle : NWCONN_HANDLE;
   NLMName : pnstr8
) : NWCCODE;
```

Parameters

connHandle

(IN) Specifies the server connection handle which is being managed.

NLMName

(IN) Points to the name of the NLM to be unloaded (NULL-terminated with a maximum length of 482 bytes).

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESS
0x009E	Bad file name or no file name given
0x0202	Unable to unload specified module
0x8801	INVALID_CONNECTION
0x89FB	ERR_NCP_NOT_SUPPORTED

0x899E	INVALID_FILENAME

Remarks

You must be logged into *connHandle*, be permanently authenticated, and have console operator rights at the minimum to call **NWSMUnloadNLM**.

The *NLMName* parameter can be any string that is valid for the UNLOAD command on the console (except it does not include the actual "UNLOAD" command) and contains the NLM name.

The *NLMName* parameter has the following format:

```
NLMname{.ext}
```

NWSMUnloadNLM does not support UNC paths.

The NLM does not have to include the extensions. .NLM will be assumed if the extension is not included.

The server console UNLOAD command is documented in the NetWare server utilities documentation.

NCP Calls

0x2222 131 2   Unload A NLM

See Also

NWSMLoadNLM

Snap-in

Snap-in: Guides

Snap-in: General Guide

Snap-in: Concepts

Snap-in: Functions

Snap-in: Structures

Parent Topic:

Management Overview

Snap-in: Tasks

Registering Object and View Snap-in DLLs Using Windows 3.1

When NetWare® Administrator loads in Windows* 3.1 it reads the NWADMN3X.INI file, located in the windows directory, to see what Snap-in DLLs need to be loaded.

1. **List all object Snap-in DLLs you want to load under "Snapin Object DLLs WIN3X".**
2. **List all view Snap-in DLLs you want to load under "Snapin View DLLs WIN3X".**

These two Snap-in entries are created automatically when you run NetWare Administrator and exit with "Save Settings on Exit" checked. For each Snap-in DLL listed in the .INI file, NetWare Administrator calls the DLL's initialization function **InitSnapin**. The DLL's initialization function must call the registration function(s) that registers each object class in the DLL. In addition, the initialization function must supply the address of a callback function for each object class.

Parent Topic:

Snap-in: General Guide

Related Topics:

General Information for Registering Object and View Snap-in DLLs

Registering Object and View Snap-in DLLs Using Windows 95

When NetWare® Administrator loads in Windows 95*, it reads the registry. Registrations for Snap-in DLLs are located in HKEY_CURRENT_USER\Software\NetWare\Parameters under NetWare Administrator. Two Snap-in entries "Snapin Object DLLs WIN95" and "Snapin View DLLs WIN95" are created automatically when you run NetWare Administrator and exit with "Save Settings on Exit" checked.

1. **To register Snap-in DLLs, select the proper key name and add a new "String Value," which contains the full path to the Snap-in DLL.**

Parent Topic:

Parent Topic:

Snap-in: General Guide

Related Topics:

General Information for Registering Object and View Snap-in DLLs

Snap-in: Concepts

Developer-Supplied Functions

As a developer you must provide the following functions and callbacks for each of your snap-in DLLs. The table shows how NetWare® Administrator uses these functions and callbacks. The prototypes for these are provided in nwsnapin.h

Table auto. Developer-supplied Functions

CanCloseSnapin	This is called by NetWare Administrator in a Snap-in DLL to determine if the Snap-in can be closed. If this call is not used NetWare Administrator assumes it is OK to shut down the DLL.
InitSnapin	NetWare Administrator calls a DLL's **InitSnapin** function immediately after loading a Snap-in DLL. The Snap-in DLL uses InitSnapin to register its Snap-in object classes and/or menu items for the "Tools" menu.
NWAFlatBrowserProc	This is a type definition for a flat browser callback proc which returns the filter values to use for the flat browser and passes the selection results of the flat browser to the DLL.
NWAMVEDProc	This is a type definition for a MVED callback proc which returns the initial values and initial values-count for the MVED control as well as the filter values for the flat browser.(if MVED values are distinguished names.)
NWASnapinMDIChildProc	This is a type definition called by NetWare Administrator to create, clear, save and restore MDI child window settings.
NWASnapinMenuActionProc	This is a type definition called by NetWare Administrator when its associated Snap-in module's menu item is selected.

NWASnapinMenuValidProc	This is a type definition called by NetWare Administrator when the user pulls down the associated Snap-in menu item.
NWASnapinTBButtonEnableProc	This is a type definition called by NetWare Administrator to find out if a toolbar button registered by a tools menu Snap-in item should be enabled or disabled.
NWASnapinObjectProc	This is a type definition for a Snap-in object class callback function. NetWare Administrator calls this with messages to initialize or close a Snap-in object class, register details pages, get a valid operation, and so on.
PostInitSnapin	This is sent by NetWare Administrator after all Snap-in DLLs have been loaded and initialized.
Shutdown	This is called by NetWare Administrator when Snap-in modules are being closed. This function must provide all Snap-in DLL-specific cleanup.

Parent Topic:

Snap-in: General Guide

General Information for Registering Object and View Snap-in DLLs

NetWare® Administrator allows you to add options in the command line at the time of execution as follows:

/F Allows you to specify a filename to read Snap-in registration information from.

This option with no parameters (no filename) acts like the default (NWADMN3X.INI for Windows 3.1 and Registry for Windows 95). For example, to save the DLLs in d:\temp\bob.txt, use the following:

```
nwadmn3x/f c:\temp\bob.txt or nwadmn95 /f c:\temp\bob.txt.
```

/N Allows you to read/save preferences to your Directory Services user object, so they can be loaded at any workstation. For example:

```
nwadmn3x /n or nwadmn95 /n
```

To have NetWare Administrator use your Snap-in DLL, you must include its name in either the NWADMN3X.INI file (for Windows 3.1) or the Registry (for Windows 95) if no option is used on the command line.

If /F (filename) is used, you must register your Snap-in DLL in the specified file name. If /N is used, Snap-in DLL REGEDT16.DLL (for Windows 3.1) or Snap-in DLL REGEDT95.DLL (for Windows 95) allows you to add or delete your Snap-in DLLs from the database.

When using the /F option, Snap-in DLLs are registered under the “Snapin Object DLLs WIN3X”, “Snapin View DLLs WIN3X”, or “Snapin Object DLLs WIN95”, “Snapin View DLLs WIN95” section as appropriate by adding a profile string similar to the following:

```
Snapex01=SNAPEX01.DLL
```

where Snapex01 is a keyword and SNAPEX01.DLL is the name of your Snap-in DLL. Make sure that keyword and Snap-in DLL names are unique.

> **IMPORTANT:** The Snap-in DLL’s shutdown function must do any DLL-specific cleanup that is required.

If a Snap-in DLL depends on information provided by another Snap-in DLL, the DLLs must be registered in the correct order. For example, an object Snap-in DLL must be registered prior to an attribute Snap-in DLL. The attribute is dependent on the object.

Parent Topic:

Snap-in: General Guide

Related Topics:

Registering Object and View Snap-in DLLs Using Windows 3.1

Registering Object and View Snap-in DLLs Using Windows 95

Multi Value Edit (MVED) Control

The MVED is a combination of the following three controls:

An edit box
A spinner
A push button

This control is used by a Snap-in object’s details window dialog. The edit control displays one value at a time; other values can be viewed by using the spinner. Values are added and deleted when the MVED value-list push-button is pushed.

The values in the MVED can be either strings or distinguished name strings. If they are distinguished-name strings, then the Flat Browser is invoked to add or delete values in the MVED.

When creating the dialog specification, use the class name “EDIT” for edit control, “microscroll” (Windows 3.1) and “microscroll32” (Windows 95) for spinner and “mvebbcontrol” for push button. Since MVED is not an MS Window’s custom control, these controls will appear as dark rectangles in you resource workshop. However, they will appear correctly in the NetWare Administrator run-time context when your Snap-in DLL is loaded.

Parent Topic:

Snap-in: General Guide

Snap-in Dialogs

NetWare® Administrator uses dialog windows to add Snap-In functionality for object classes. These dialogs are displayed in an objects’s details window or in the object’s MPEW window. A Snap-in DLL can provide dialogs for a new Snap-in object’s details. The Snap-in DLL can also enhance the functionality of an existing object by adding additional dialogs for the object.

Parent Topic:

Snap-in: General Guide

Snap-in DLL Functions

Snap-In functions are provided by compiler-specific SNAPIN3X.LIB and SNAPIN95.LIB files, with the associated header file NWSNAPIN.H. The following functions are part of Snap-In.

NWAAddClassData	Adds the translated class name and browser bitmaps to the NetWare® Administrator tables.
NWAAddPropertyNameTranslation	Adds the translated property names to the NetWare Administrator translation tables.
NWACreateMDIChildWindow	Creates an instance of an MDI child window in which a Snap-in view window can be created.
NWACreateMved	Creates and initializes Multi Value Edit (MVED) control in a dialog box.
NWACreateWindowMenu	Inserts a Window drop-down menu at the specified location in a specified Snap-in view menu.
NWAExitNWAdmin	Shuts down NetWare

	Administrator.
NWAGetClassAliasBitmap	Returns the alias bitmap for any object class registered in NetWare Administrator.
NWAGetClassBitmap	Returns the class bitmap for any object class registered in NetWare Administrator.
NWAGetClassReadOnlyBitmap	Returns the read-only bitmap for any object class registered in NetWare Administrator.
NWAGetConsoleWindowHandle	Returns the window handle of NetWare Administrator Console Window.
NWAGetMvedCount	Returns the count of values in an MVED control.
NWAGetMvedValue	Returns a string value from an MVED control.
NWAGetNLSFilePath	Retrieves the path for the specified help/resource file in the NetWare Administrator environment.
NWAGetNWAdminVersion	Returns the NetWare Administrator version number.
NWAGetSaveSettingsOption	Returns the value of the "Save Settings on Exit" option in the NetWare Administrator menu.
NWAGetSelObject	Iteratively returns the objects selected in the active Browser.
NWAGetSelObjectCount	Returns the count of the objects selected in the active Browser.
NWAGetToolsMenuItem	Gets a Tools menu item present under the Tools drop-down menu in a specified view class menu.
NWAGetToolsMenuItemCount	Returns the number of menu items present under the Tools drop-down menu in a specified view class menu.
NWAGetTranslatedClassName	Returns the translated name for any object class in NetWare Administrator.
NWAGetTranslatedPropertyName	Returns the translated property name for any object property in NetWare Administrator.
NWAGetTreeName	Gets a tree name and a browser context corresponding to an active

	browser window in NetWare Administrator.
NWALaunchConfigDialog	Launches a dialog box to configure the toolbar and status bar preferences.
NWALaunchDetails	Launches a details window.
NWALaunchDSFlatBrowser	Launches a DS Flat Browser.
NWALaunchFSFlatBrowser	Launches an FS Flat Browser.
NWAProcessToolsMenuItemCommand	Executes a command corresponding to a specified Tools menu item.
NWAProcessToolsMenuItemValid	Executes a command to enable or disable a Tools menu item.
NWARegisterMDIChildWindow	Registers an MDI child window class name with NetWare Administrator.
NWARegisterMenu	Registers menu items when they are added to the Tools menu and registers the functions to be called when the Snap-in menu item is selected.
NWARegisterObjectProc	Registers an object class for the Snap-in module.
NWARegisterObjectProcEx	Registers an object class for the Snap-in module (extended version).
NWARegisterToolBarButton	Allows tools that Snap-in to the NetWare Administrator Tools menu to register a button to be displayed on the toolbar.
NWARemoveClassData	Removes all class bitmaps from NetWare Administrator bitmap tables.
NWARemovePropertyNameTranslation	Removes the translated property name for any property in NetWare Administrator.

Parent Topic:

Snap-in: General Guide

Snap-in DLL Menu Item Registering

Snap-in DLLs can register menu items to be added to NetWare®

Administrator's "Tools" menu. Snap-in DLLs must register two callbacks for each menu item: one callback enables/disables the menu item, and the other responds when the item is selected.

Parent Topic:

Snap-in: General Guide

Snap-in Messages Sent by NetWare Administrator

The Snap-in interface requires that the Snap-in module developer register callback functions to respond to certain messages (see the next section). These messages are described below:

NWA_MSG_APPLYTEMPLATE	This is sent to the Snap-in(s) with the object name being created and the template object name being used to create the object.
NWA_MSG_CLOSESNAPIN	This is the last message a Snap-in receives and it should unregister its translated names and bitmaps, and free allocated memory.
NWA _MSG_CREATEOBJECT	This is sent by NetWare® Administrator to the Snap-in when a user selects the object class supported by the Snap-in from the Create dialog in the Object menu.
NWA_MSG_FBFILTER_COUNT	This is the first message a Flat Browser or MVED callback function receives.
NWA_MSG_FBFILTER_VALUE	This is sent to the Flat Browser or MVED callback function with the selected filter index.
NWA_MSG_FBOBJECT_COUNT	This is sent to the Flat Browser callback function with the count of selected objects in the flat browser, if the OK button is pressed.

NWA_MSG_FBOBJECT_VALUE	This is sent to the Flat Browser callback function with the selected object index.
NWA_MSG_GETPAGECOUNT	This is sent by NetWare Administrator to the Snap-in when a user selects the object supported by the Snap-in and then selects Details from the Object menu.
NWA_MSG_GETVALIDOPERATIONS	This is sent by NetWare Administrator to **SnapinObjectProc** function when a Snap-in object is selected and the user selects a menu operation.
NWA_MSG_INITSNAPIN	This is the first message a Snap-in receives indicating it should register bitmaps, translated property, and class members.
NWA_MSG_MDICHILD_CLEARSETTIN GS	This is sent to the **NWASnapinMDIChildP roc** callback function by NetWare Administrator when NetWare Administrator is going down and the Save settings option in NetWare Administrator is set.
NWA_MSG_MDICHILD_CREATED	This is sent to the **NWASnapinMDIChildP roc** callback function by NetWare Administrator when an instance of the MDI child window is created.
NWA_MSG_MDICHILD_RESTORE	This is sent to the **NWASnapinMDIChildP roc** callback function by NetWare Administrator when NetWare Administrator comes up.
NWA_MSG_MDICHILD_SAVESETTING S	This is sent to the **NWASnapinMDIChildP**

	roc callback function by NetWare Administrator when NetWare Administrator is going down and the Save settings option in NetWare Administrator is set.
NWA_MSG_MODIFY	This is sent by NetWare Administrator to the Snap-in when the user selects the OK button from the MPEW after making modifications.
NWA_MSG_MPEWCLOSE	This is sent by NetWare Administrator to the Snap-in when the details window (MPEW) is being closed.
NWA_MSG_MULTIOBJ_COUNT	This is sent to the **NWASnapinObjectProc** callback function indicating the number of objects selected in the browser to perform multiple object details.
NWA_MSG_MULTIOBJ_NAME	This is sent to the **NWASnapinObjectProc** callback function with the complete DN name of the selected object.
NWA_MSG_MVED_INITCOUNT	This is the first message an MVED callback function receives indicating the number of MVED values to initialize.
NWA_MSG_MVED_INITVALUE	This is sent to the MVED callback function with an index of the MVED value.
NWA_MSG_NOTIFYCREATEOBJECT	This is sent by NetWare Administrator once the object is created.
NWA_MSG_NOTIFYDELETEOBJECT	NetWare Administrator sends this notification message to Snap-in when the object is deleted.
NWA_MSG_NOTIFYMOVE	NetWare Administrator

	sends this notification message to Snap-in when the object is moved.
NWA_MSG_NOTIFYRENAME	NetWare Administrator sends this notification message to Snap-in when the object is renamed.
NWA_MSG_QUERYCOPY	This is sent by NetWare Administrator to the Snap-in when the user selects an object that is supported by the Snap-in and then selects Copy from the Object menu.
NWA_MSG_QUERYDELETE	This is sent by NetWare Administrator to the Snap-in when the user selects an object that is supported by the Snap-in and then selects Delete from the Object menu.
NWA_MSG_QUERYMOVE	This is sent by NetWare Administrator to the Snap-in when the user selects an object that is supported by the Snap-in and then selects Move from the Object menu.
NWA_MSG_REGISTERPAGE	This is sent by NetWare Administrator to the Snap-in “n” times, where “n” is the number returned by the NWA_MSG_GETPAGECOUNT message.
NWA_MSG_RENAME	This is sent by NetWare Administrator to the Snap-in when a user selects the object supported by the Snap-in and then selects Rename from the Object menu.
NWA_MSG_STATUSBAR_ADDPREFITEM	This is sent to the MDI child window by the NetWare Administrator main window to the number returned by the NWA_MSG_STATUSBAR_QUERYITEMCOUNT

	message.
NWA_MSG_STATUSBAR_DBLCLK	This is sent to an MDI child window when the user double-clicks the left mouse button over a field in the status bar.
NWA_MSG_STATUSBAR_NEWACTIVEITEMS	This is sent to a view by the NetWare Administrator main window when a user has pressed the OK button in the status bar preferences dialog box.
NWA_MSG_STATUSBAR_POPULATE	This is sent to the top MDI child window after the NetWare Administrator main window has handled a NWA_MSG_IVEGOTFOCUS message.
NWA_MSG_STATUSBAR_QUERYACTIVECOUNT	This is sent to an MDI child window when the user opens the status bar preferences dialog box.
NWA_MSG_STATUSBAR_QUERYACTIVEITEMS	This is sent to an MDI child window when the user opens the status bar preferences dialog box and has responded to a NWA_MSG_STATUSBAR_QUERYACTIVECOUNT message.
NWA_MSG_STATUSBAR_QUERYITEMCOUNT	This is sent to an MDI child window when the user opens the status bar preferences dialog box.
NWA_MSG_TOOLBAR_ADDPREFITEM	This is sent by the NetWare Administrator main window to the view window according to the number returned by the NWA_MSG_TOOLBAR_QUERYITEMCOUNT message.
NWA_MSG_TOOLBAR_NEWACTIVEITEMS	This is sent from the NetWare Administrator main window to a view when the user has pressed OK in the toolbar

	preferences dialog box.
NWA_MSG_TOOLBAR_POPULATE	This is sent to the top MDI child window each time the toolbar is created.
NWA_MSG_TOOLBAR_QUERYACTIVE COUNT	This is sent to an MDI child window when the user opens the toolbar preferences dialog box.
NWA_MSG_TOOLBAR_QUERYACTIVEI TEMS	This is sent to a view when the user opens the toolbar preferences dialog box after the view responds to a NWA_MSG_TOOLBAR_ QUERYACTIVECOUNT message.
NWA_MSG_TOOLBAR_QUERYENABLE BUTTON	This is sent to an MDI child window when the NetWare Administrator main window wants to know if a toolbar item should be enabled or disabled (grayed out).
NWA_MSG_TOOLBAR_QUERYITEMCO UNT	This is sent to an MDI child window when the user opens the toolbar preferences dialog box.
NWA_WM_CANCLOSE	Receives this message when OK is selected and is the dialog procedure for a page dialog in a MPEW.
NWA_WM_F1HELP	Receives this message when F1 Help) is selected and is the dialog procedure for a page dialog in MPEW.

Parent Topic:

Snap-in: General Guide

Related Topics:

Snap-in Messages Sent by the Snap-in

Snap-in Messages Sent by the Snap-in

NWA_MSG_IVEGOTFOCUS	This is sent by an MDI child window to the NetWare® Administrator main window when the child window receives a WN_SETFOCUS message.
NWA_MSG_STATUSBAR_ADDITEM	This is sent to the NetWare Administrator main window after it sends a NWA_MSG_STATUSBAR_POPULATE message to the top MDI child window.
NWA_MSG_STATUSBAR_SETITEMTEXT	This replaces the text in a previously defined status bar text field.
NWA_MSG_TOOLBAR_ADDITEM	This is sent to the top MDI child window after it sends a NWA_MSG_TOOLBAR_POPULATE message to the NetWare Administrator main window.
NWA_MSG_TOOLBAR_GETBUTTONSTATE	This is sent from an MDI child window to the NetWare Administrator main window to determine the state of the button corresponding to the button ID.
NWA_MSG_TOOLBAR_GETBUTTONTYPE	This is sent by a view to the NetWare Administrator main window to determine what type of button corresponds to the button ID.
NWA_MSG_TOOLBAR_SETBUTTONSTATE	This is sent by an MDI child window to the NetWare Administrator main window to set the state of the button corresponding to the button ID.
NWA_WM_SETPAGEMODIFY	This should be sent to the page dialog window to indicate the page dialog information has changed.

Parent Topic:

Snap-in: General Guide

Related Topics:

Snap-in Messages Sent by NetWare Administrator

Snap-in Introduction

Snap-in provides a “Snap-in” interface that allows developers to snap-in objects into the NetWare® Administrator console. Snap-in allows a developer’s applications to do the following:

Add a menu item under the “Tools” menu in the NetWare Administrator console.

Enhance the details of an existing object class by registering additional pages to the details Multi Page Edit Widget (MPEW) window.

Add details for a new Snap-in object by registering MPEW pages.

Add bitmaps (object bitmap, alias bitmap, and read-only bitmap) and translation for object class names and object class property names.

Register Snap-in object classes and menu items with NetWare Administrator.

Invoke the DS Flat Browser and File System Flat Browser in the developer’s Snap-in dynamic link library (DLL) to browse Novell® Directory Services™ (NDS™) and file system objects.

Create a Multi Values Edit (MVED) control in Snap-in dialogs that implement MPEW pages. (However, it is not possible to use this control in any other window or dialog as the control data is being managed by the MPEW window.) The interfaces for the MVED and Flat Browser enable the developer’s DLL to have a user interface similar to NetWare Administrator.

Register a Snap-in view class and its associated menu, icon, and toolbar and status bar items.

Create an instance of a Snap-in view window.

A Snapin-view window is an MDI child window. When the parent window becomes active, the child window’s menu, toolbar and status bar are displayed in NetWare Administrator.

Parent Topic:

Snap-in: General Guide

Snap-in Status Bar

The Snap-in interface allows the MDI child window to manage the entire contents of the NWAdmin Status Bar. It can add items, change the text of those items and respond to user mouse double-clicks on an item.

Parent Topic:

Snap-in: General Guide

Related Topics:

Snap-in View

Snap-in Toolbar

The Snap-in interface allows a Snap-in to add buttons to the NWAdmin Main Window Toolbar when the Snap-in MDI Child Window becomes active. The MDI child window shares the toolbar with the NWAdmin main window and all the child's buttons display contiguously (with possible interspersed separators) in a position determined by the user.

Parent Topic:

Snap-in: General Guide

Related Topics:

Snap-in View

Snap-in View

When you register a Snap-in view with NetWare® Administrator a view proc, menu, icon and, if desired, toolbar and status bar items are also registered. Registration is done using InitSnapin, where all other Snap-in components are registered.

It is recommended that a menu item be added to NetWare Administrator's Tools menu where the view can be opened. The Snap-in view window opens as an MDI child. When the MDI child window becomes active, its menu displays in NetWare Administrator's frame window. When the MDI child is minimized, its corresponding icon displays. The toolbar and status bar is updated to correspond with the active view.

It is recommended that a Snap-in view also contain a Tools menu that is populated with the same menu items found in NetWare Administrator's Tools menu. This way, any instance of a view can be invoked from any other instance.

Parent Topic:

Snap-in: General Guide

Related Topics:

Snap-in Status Bar

Snap-in Toolbar

The Flat Browser

NetWare® Administrator uses the Flat Browser to browse NDS objects and file system objects. The Flat Browser displays two lists:

The list on the right is a container list and displays container objects.
The list on the left displays objects contained by the selected object in the container list.

The Flat Browser also allows the user to filter displayed objects.

Parent Topic:

Snap-in: General Guide

The Snap-in Interface

A Snap-in module is implemented as one or more dynamic-link libraries (DLLs) with a message-passing callback mechanism. A single DLL can have Snap-in functionality for one or more object classes. A typical developer's Snap-in DLL consists of the components listed in the table below.

Table auto. Components Typically Used in a DLL

Initialization function	The initialization function registers the Snap-in DLL's object classes, "Tools" menu item(s), and view classes.
Shutdown function	The shutdown function is called when the Snap-in modules are being closed. If it is present, **CanCloseSnapin** is called before shut down takes place. If all snap-ins respond affirmatively, then NetWare® Administrator will proceed with normal shut down.
Callback function(s)	NetWare Administrator sends messages to the Snap-in object's callback function when the user requests an action on the object for which the Snap-in object is responsible. One callback exists for each object class.
Snap-in dialogs	These dialogs provide the Snap-in functionality for object classes, and are displayed when an object's details are invoked. Dialogs can be created to administer the details of a new object class or to enhance the details of an existing object class.
Snap-in menu items	These menu items are added to the "Tools" menu of NetWare Administrator. For each menu item that is registered, a "valid" and an "action" callback function

	must be registered. The “valid” callback is invoked when the Tools menu is pulled down. (This is where the menu item is enabled and disabled.) The “action” callback is invoked when the menu item is selected.
Snap-in view	A Snap-in view registers a view class, proc, menu and icon with NetWare Administrator.

Parent Topic:

Snap-in: General Guide

Snap-in: Functions

CanCloseSnapin

Is called by NetWare Administrator in a Snap-in DLL to determine if the Snap-in can be closed

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nbool FAR PASCAL CanCloseSnapin
   (void);
```

Return Values

TRUE
FALSE

Remarks

CanCloseSnapin returns TRUE if Snap-in is ready to be closed. Otherwise, FALSE will be returned.

If FALSE is returned, NetWare Administrator will display an appropriate message explaining that the Snap-in is not ready to close and still allow you to continue with the shut down (even though any unsaved information might be lost).

Snap-in does not have to implement **CanCloseSnapin**. NetWare Administrator will assume that the Snap-in is ready to close if **CanCloseSnapin** is not available in a Snap-in DLL.

NCP Calls

None

InitSnapin

Is called by NetWare Administrator when it loads the Snap-in DLL

Syntax

```
#include <nwsnapin.h>
```

Return Values

NWA_RET_SUCCESS

Remarks

When **InitSnapin** is called, it must register the DLL's Snap-in object classes, view classes, and/or menu items for the tools menu.

See Also

ShutDown

NWAAddClassData

Adds the translated class name and browser bitmaps to the NetWare Administrator tables

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAAddClassData
   (LPCSTR    pszClassName,
    LPCSTR    pszTranslation,
    HBITMAP   hClassBitmap,
    HBITMAP   hClassAliasBitmap,
    HBITMAP   hClassReadOnlyBitmap);
```

Parameters

pszClassName

(IN) Specifies the schema class name.

pszTranslation

(IN) Specifies the translated name.

hClassBitmap

(IN) Specifies the normal bitmap of the class.

hClassAliasBitmap

(IN) Specifies the alias bitmap of the class.

hClassReadOnlyBitmap

(IN) Specifies the read only bitmap of the class.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_DUPLICATE_ENTRY

Remarks

The Snap-in calls **NWAAddClassData** when it receives NWA_MSG_INITSNAPIN. NetWare Administrator uses these bitmap parameters to display objects of this type.

The *hClassBitmap* and *hClassReadOnlyBitmap* must be 16x16 pixels in size and *hClassAliasBitmap* must be 24x16 pixels in size.

NCP Calls

None

See Also

NWA_MSG_INITSNAPIN, NWA_MSG_CLOSESNAPIN,
NWAGetClassBitmap, **NWAGetClassAliasBitmap**,
NWAGetClassReadOnlyBitmap, **NWARemoveClassData**

NWAAddPropertyNameTranslation

Adds the translated property names to the NetWare Administrator translation tables

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAAddPropertyNameTranslation
   (LPCSTR   pszPropertyName,
    LPCSTR   pszClassName,
    LPCSTR   pszTranslation);
```

Parameters

pszPropertyName

(IN) Specifies the schema property name.

pszClassName

(IN) Specifies the schema class name.

pszTranslation

(IN) Specifies the translated property name.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_ATTRNAME_NOT_FOUND

Remarks

If *pszClassName* is NULL, **NWAAddPropertyNameTranslation** creates a default translation for the property. The default translation is used if other translations are not specified by a class name. To remove default translations, call **NWARemovePropertyNameTranslation**, passing NULL to *pszClassName*.

NCP Calls

None

See Also

NWAGetTranslatedPropertyName, NWARemovePropertyNameTranslation

NWACreateMDIChildWindow

Creates an instance of an MDI child window in which a Snap-in view window can be created

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWACreateMDIChildWindow
  (LPCSTR                   className,
   LPCSTR                   windowTitle,
   HWND                    *pHwnd,
   NWACreateMDIChildParam  *pCreateParam);
```

Parameters

className

(IN) Specifies the registered view name to be created.

windowTitle

(IN) Specifies the view title to be displayed on view window.

pHwnd

(OUT) Points to the MDI child window handle created.

pCreateParam

(IN) Points to the window position parameters which are used to create the MDI child window.

Return Values

NWA_RET_SUCCESS
NWA_ERR_NO_MEMORY
NWA_ERR_CLASSNAME_NOT_FOUND

Remarks

If the view name specified by *className* is not registered before calling **NWACreateMDIChildWindow**, an error will be returned.

Snap-in must specify *pCreateParam* when the MDI child window is being restored to the last position in which in was displayed; otherwise, it might be set to NULL.

NCP Calls

NCP Calls

None

See Also

NWARegisterMDIChildWindow

NWACreateMved

Creates and initializes Multi Value Edit (MVED) control in a dialog box

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWACreateMVED
  (HWND           hwndParent,
   nuint32        userParam,
   nuint16        idEdit,
   nuint16        idSpin,
   nuint16        idButton,
   nuint16        editType,
   nuint16        lengthEdit,
   LPCSTR         label,
   LPCSTR         contextStr,
   nuint32        brwsFlags,
   NWAMVEDProc    pfn);
```

Parameters

hwndParent

(IN) Specifies the parent window handle of MVED control. *hwndParent* is the handle of the Snap-in dialog that has MVED control.

userParam

(IN) Specifies the user parameter that will be provided in MVED callback functions.

idEdit

(IN) Specifies the dialog control ID of MVED edit control.

idSpin

(IN) Specifies the dialog control ID of MVED spin control.

idButton

(IN) Specifies the dialog control ID of MVED push button control.

editType

(IN) Specifies the MVED edit type.

lengthEdit

(IN) Specifies the length of the MVED edit field. Its length must be MWA_MAX_OBJECT_NAME if *editType* is NWA_MVED_VALUE_DISTNAME.

label

(IN) Specifies the label used in add/delete string dialog which is displayed when MVED push button is pressed.

contextStr

(IN) Specifies the context string used for Flat Browser display.

brwsFlags

(IN) is currently not used and must be zero.

pfn

(IN) Specifies the MVED callback function. *pfn* is invoked with the messages to initialize MVED control. It is also used to specify object class filters for Flat Browser display if *editType* is NWA_MVED_VALUE_DISTNAME.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_NO_MEMORY

Remarks

MVED control is a combination of three controls, i.e. edit control, spin control, and push button control. Display coordinates must be changed to specify correct position on the dialog.

If dialog is viewed in resource work-shop, controls will be displayed as dark rectangles; and they will appear fine in NetWare Administrator run time environment.

editType can have one of the following types:

NWA_MVED_VALUE_READONLY allows MVED control to be read only and is used for only displaying values. If this type is used, push button control is not necessary. *contextStr* and *brwsFlags* will be ignored with this type.

NWA_MVED_VALUE_STRING allows MVED control of string values. If MVED push button is pressed to add/delete values, the dialog to add/delete simple string values will be displayed. *contextStr* and *brwsFlags* will be ignored with this type.

NWA_MVED_VALUE_DISTNAME allows MVED control over distinguished name string values. If MVED push button is pressed to add/delete values, the Flat Browser to add/delete distinguished name string values will be displayed.

NWA_MVED_VALUE_DIGITSONLY allows MVED control over digit-only string values. If MVED push button is pressed to add/delete values, the dialog to add/delete digit-only string values will be displayed. *contextStr* and *brwsFlags* will be ignored with this type.

NWA_MVED_VALUE_PRINTABLE allows MVED control over printable string values. If MVED push button is pressed to add/delete values, the dialog to add/delete printable string values will be displayed. *contextStr* and *brwsFlags* will be ignored with this type.

If MVED is implemented in a dialog box which is managed by a message-mapping system (such as OWL and MFC class libraries), the WM_VSCROLL messages must be passed to the default dialog box procedure.

NCP Calls

None

See Also

NWAGetMvedCount, **NWAGetMvedValue**

NWACreateWindowMenu

Inserts a Window drop-down menu at the specified location in a specified Snap-in view menu

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWACreateWindowMenu
   (pnstr8   className,
    HMENU    hMenu,
    uint     menuPos);
```

Parameters

className

(IN) Points to the MDI child window class name in which the window menu will be created.

hMenu

(IN) Specifies the handle of the menu where the Window drop-down menu will be inserted.

menuPos

(IN) Specifies the zero-based position at which the drop-down menu will be inserted.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_INVALID_PARAMETER

Remarks

Snap-in can call **NWACreateWindowMenu** to insert a Window drop-down menu at a given position in an MDI child window specific menu.

The inserted window menu can not be added in the first or last position. There must be at least 2 top-level items in*hMenu* prior to calling **NWACreateWindowMenu**.

NCP Calls

Management Service Group

None

NWAExitNWAdmin

Shuts down NetWare Administrator
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY (NWRCODE) NWAExitNWAdmin
   (void);
```

Return Values

None

Remarks

NWAExitNWAdmin closes all the active views and detail windows (if present). It also closes and performs Shutdown on the Snap-in(s) loaded in the system.

NCP Calls

None

NWAFlatBrowserProc

Is a type definition for a flat browser callback proc which returns the filter values to use for the flat browser and passes the selection results of the flat browser to the DLL

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK (nuint32, NWAFlatBrowserProc)
   (nuint32   userParam,
    nuint16   uMsg,
    nparam    param1,
    nparam    param2);
```

Parameters

userParam

Specifies the User Param provided in Launch Flat Browser function.

uMsg

Specifies the message to the callback.

param1

Specifies parameter 1 of the message.

param2

Specifies parameter 2 of the message.

Messages

NWA_MSG_FBFILTER_COUNT
NWA_MSG_FBFILTER_VALUE
NWA_MSG_FBOBJECT_COUNT
NWA_MSG_FBOBJECT_VALUE

Remarks

FlatBrowserProc is provided as a parameter to **NWALaunchFSFlatBrowser** and **NWALaunchDSFlatBrowser**. **FlatBrowserProc** is invoked to return the filter values to use for Flat Browser. The selection results of the Flat Browser are passed to the DLL through this callback.

NCP Calls

None

See Also

NWALaunchDSFlatBrowser, **NWALaunchFSFlatBrowser**

NWAGetClassAliasBitmap

Returns the alias bitmap for any object class registered in NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetClassAliasBitmap
  (LPCSTR      pszClassName,
   HBITMAP    *phBitmap);
```

Parameters

pszClassName

(IN) Specifies the schema class name.

phBitmap

(OUT) Points to the class alias bitmap.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_BITMAP_NOT_FOUND

Remarks

If successful, *phBitmap* points to the class alias bitmap. If the class alias bitmap is not found, NWA_ERR_BITMAP_NOT_FOUND is returned. Otherwise, NWA_ERR_ERROR is returned.

NCP Calls

None

See Also

NWAGetClassBitmap, **NWAGetClassReadOnlyBitmap**

NWAGetClassBitmap

Returns the class bitmap for any object class registered in NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetClassBitmap
   (LPCSTR      pszClassName,
    HBITMAP    *hBitmap);
```

Parameters

pszClassName

(IN) Specifies the schema class name.

phBitmap

(OUT) Points to the class bitmap.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_BITMAP_NOT_FOUND

Remarks

If successful, *phBitmap* points to the class bitmap. If the class bitmap is not found, NWA_ERR_BITMAP_NOT_FOUND is returned. Otherwise, NWA_ERR_ERROR is returned.

NCP Calls

None

See Also

NWAGetClassReadOnlyBitmap, **NWAGetClassAliasBitmap**

NWAGetClassReadOnlyBitmap

Returns the read-only bitmap for any object class registered in NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetClassReadOnlyBitmap
   (LPCSTR     pszClassName,
    HBITMAP   *phBitmap);
```

Parameters

pszClassName

(IN) Specifies the schema class name.

phBitmap

(OUT) Points to the class read only bitmap.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_BITMAP_NOT_FOUND

Remarks

If successful, *phBitmap* points to the class read only bitmap. If the class read only bitmap is not found, NWA_ERR_BITMAP_NOT_FOUND is returned. Otherwise, NWA_ERR_ERROR is returned.

NCP Calls

None

See Also

NWAGetClassBitmap, **NWAGetClassAliasBitmap**

NWAGetConsoleWindowHandle

Returns the window handle of NetWare Administrator Console Window

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(HWND) NWAGetConsoleWindowHandle
  (void);
```

Return Values

HWND: Window Handle of NetWare Administrator Console Window

Remarks

The parent window of the console window is the window to which Snap-in views should send view-related messages.

NCP Calls

None

See Also

NWAGetMvedValue, NWACreateMved

NWAGetMvedCount

Returns the count of values in an MVED control

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetMvedCount
   (hwnd       hwndParent,
    nuint      idEdit,
    pnuint16   pCount);
```

Parameters

hwndParent

(IN) Specifies the parent window handle of MVED control. It is the handle of the dialog that contains MVED control.

idEdit

(IN) Specifies the dialog control ID of MVED edit control.

pCount

(OUT) Points to an integer.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

pCount points to the count of values MVED control has if **NWAGetMvedCount** returns NWA_RET_SUCCESS.

In order to receive the values present in MVED control, first get the count of the values and then iterate over MVED control by calling **NWAGetMvedValue**.

NCP Calls

None

See Also

NWAGetMvedValue, NWACreateMved

NWAGetMvedValue

Returns a string value from an Mved control

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetMvedValue
   (hwnd      hwndParent,
    nuint16   idEdit,
    nuint16   itemIndex,
    nuint16   bufSize,
    pnstr     pValue);
```

Parameters

hwndParent

(IN) Specifies the parent window handle of MVED control. It is the handle of the dialog that contains MVED control.

idEdit

(IN) Specifies the dialog control ID of MVED edit control.

itemIndex

(IN) Specifies the index of the string in MVED control.

bufSize

(IN) Specifies the size of the buffer.

pValue

(OUT) Points to a buffer in which a value will be placed.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If **NWAGetMvedValue** returns NWA_RET_SUCCESS, *pValue* will have a value present at the *itemIndex* position in the MVED control.

NCP Calls

None

See Also

NWAGetMvedCount, **NWACreateMved**

NWAGetNLSFilePath

Retrieves the path for the specified help/resource file in the NetWare Administrator environment

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetNLSFilePath
   (pnstr8   fileName,
    pnstr8   pathName);
```

Parameters

fileName

(IN) Points to the file name for which to search.

pathName

(OUT) Points to the path where the specified file is located.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_PARAMETER
NWA_ERR_FILE_NOT_FOUND
NWA_ERR_INSUFFICIENT_MEMORY

Remarks

NWAGetNLSFilePath searches for the specified file (resource or help) in the appropriate paths specified by the NWLANGUAGE environment variable.

Snap-in should allocate the buffer pointed to by *szPathName* to be at least NWA_MAX_PATH_SIZE.

NWA_MAX_PATH_SIZE includes the drive letter in its path. **NWAGetNLSFilePath** first searches for the file in the NLS directory under the load area (from wherever NetWare Administrator is loaded). It then searches the load area itself. If the file is still not found,*szPathName* will contain an empty string value (“”).

NetWare Administrator uses the same algorithm to find a help or resource file. Snap-in can use the same algorithm by calling **NWAGetNLSFilePath**.

Snap-in should localize the help and resource file text and install the help file in an appropriate language specific sub-directory (specified by NWLANGUAGE) in the NLS directory under the NetWare Administrator load area.

NCP Calls

None

NWAGetNWAdminVersion

Returns the NetWare Administrator version number
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetNWAdminVersion
  (pnuint32   pVersion);
```

Parameters

pVersion
(OUT) Points to the NetWare Administrator version number.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If successful, *pVersion* points to 32 bit version information packed as:

HIGH WORD

HIGH BYTE: NW Admin Major Version
LOW BYTE: NW Admin Minor Version

LOW WORD

HIGH BYTE: 0
LOW BYTE: NW Admin Revision Number

NCP Calls

None

NWAGetSaveSettingsOption

Returns the value of the "Save Settings on Exit" option in the NetWare Administrator menu

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetSaveSettingsOption
  (BOOL  *pOptionFlag);
```

Parameters

pOptionFlag

(OUT) Points to the save settings option flag.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_PARAMETER

Remarks

NWAGetSaveSettingsOption may be called to determine if the Save Settings option is set. If the option is set, Snap-in should save settings.

NCP Calls

None

See Also

NWACreateMDIChildWindow

NWAGetSelObject

Iteratively returns the objects selected in the active Browser
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetSelObject
   (nint32          index,
    NWASelObject   selObject);
```

Parameters

index
(IN) Specifies the index of the selected object.

selObject
(OUT) Points to the selected object.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

NetWare Administrator returns the requested object information through *selObject*.

index can be any value between 0 and *lpCount*-1, where *lpCount* is returned by **NWAGetSelObjectCount**.

NCP Calls

None

See Also

NWAGetSelObjectCount

NWAGetSelObjectCount

Returns the count of the objects selected in the active Browser

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetSelObjectCount
  (pnint32   lpCount);
```

Parameters

lpCount

(OUT) Points to the object count value.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

NetWare Administrator returns the object count through*lpCount*.

NCP Calls

None

See Also

NWAGetSelObject

NWAGetToolsMenuItem

Gets a Tools menu item present under the Tools drop-down menu in a specified view class menu

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetToolsMenuItem
  (pnstr8              viewName,
   nuint16             itemIndex,
   NWAToolsMenuItem*   pMenuItem);
```

Parameters

viewName

(IN) Points to the name of the view, NetWare Administrator console, or MDI child window class name under which the Tools menu items are registered.

itemIndex

(IN) Specifies the zero-based index of the items under the Tools drop-down menu.

pMenuItem

(OUT) Points to the NWAToolsMenuItem structure in which the result will be placed.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_CLASSNAME
NWA_ERR_INVALID_PARAMETER
NWA_ERR_CLASSNAME_NOT_FOUND
NWA_ERR_ITEM_NOT_FOUND

Remarks

If *viewName* is set to NWA_VIEW_CONSOLE, the information about the menu item under the Tools drop-down menu in the NetWare Administrator console window will be provided.

If *viewName* is set to NWA_VIEW_BROWSER, the information about the menu item under the Tools drop-down menu in the NetWare

Administrator browser view will be provided.

If NWA_RET_SUCCESS is returned by **NWAGetToolsMenuItem**, *pMenuItem* receives a menu item at the index specified by *itemIndex*.

You should provide a Tools drop-down menu in a Snap-in MDI child window menu. To make NetWare Administrator Tools functionality available to the Snap-in MDI child window, call **NWAGetToolsMenuItem** to get all the Tools menu items and add items using the Snap-in Tools drop-down menu.

The NWAToolsMenuItem structure returns requested menu item information.

NCP Calls

None

See Also

NWAGetToolsMenuItemCount, **NWAProcessToolsMenuItemCommand**

NWAGetToolsMenuItemCount

Returns the number of menu items present under the Tools drop-down menu in a specified view class menu

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetToolsMenuItemCount
   (pnstr8      viewName,
    pnuint16    pCount);
```

Parameters

viewName

(IN) Points to the name of the View, NetWare Administrator console, or MDI child window class name under which the Tools menu items are registered:

NWA_VIEW_CONSOLE
NWA_VIEW_BROWSER

pCount

(OUT) Points to a nuint16 value (non-NULL) where the menu item count will be returned.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_PARAMETER
NWA_ERR_INVALID_CLASSNAME
NWA_ERR_CLASSNAME_NOT_FOUND

Remarks

If *viewName* is set to NWA_VIEW_CONSOLE, the count of menu items under the Tools drop-down menu in the NetWare Administrator console window will be provided.

If *viewName* is set to NWA_VIEW_BROWSER, the count of menu items under the Tools drop-down menu in the NetWare Administrator browser view will be provided.

pCount points to the count of menu items under the Tools drop-down menu if NWA_RET_SUCCESS is returned.

If *pCount* contains a NULL pointer, **NWAGetToolsMenuItemCount** will return an error.

The Tools drop-down menu is provided in the NetWare Administrator console and the NetWare Administrator browser window.

Although the Snap-in MDI child window (which implements some view of the Snap-in data) is allowed to provide its own MDI child window class specific menu, it is recommended that Snap-in provides the Tools menu and implements the functionality of the Tools menu under the NetWare Administrator console. Snap-in might also add some functionality from the Tools menu registered under other views or MDI child window classes.

To receive information about a menu item, get the count of the menu items and iterate by calling **NWAGetToolsMenuItem**.

NCP Calls

None

See Also

NWAGetToolsMenuItem, **NWAProcessToolsMenuItemCommand**

NWAGetTranslatedClassName

Returns the translated name for any object class in NetWare Administrator
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetTranslatedClassName
   (LPCSTR   pszClassName,
    pnstr8   pszTranslation,
    nint16   nMaxLen);
```

Parameters

pszClassName

(IN) Specifies the schema class name.

pszTranslation

(OUT) Points to a buffer for the translated name.

nMaxLen

(IN) Specifies the length of the buffer for the translated name.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_ERR_INSUFFICIENT_STRING_BUFFER
NWA_ERR_CLASSNAME_NOT_FOUND

Remarks

If translation is found and copied into *pszTranslation* up to a length of *nMaxLen*, **NWAGetTranslatedClassName** returns NWA_RET_SUCCESS; if there is an error, **NWAGetTranslatedClassName** returns NWA_ERR_ERROR. If *nMaxLen* is not of sufficient length, **NWAGetTranslatedClassName** returns NWA_ERR_INSUFFICIENT_STRING_BUFFER.

The buffer of length *nMaxLen* should include two extra bytes for the NULL character (instead of one byte) to accomodate internationalized strings.

NCP Calls

None

See Also

NWAAddClassData, **NWARemoveClassData**

NWAGetTranslatedPropertyName

Returns the translated property name for any object property in NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetTranslatedPropertyName
   (LPCSTR   pszPropertyName,
    LPCSTR   pszClassName,
    pnstr8   pszTranslation,
    nint16   nMaxLen);
```

Parameters

pszPropertyName

(IN) Specifies the schema property name.

pszClassName

(IN) Specifies the schema class name.

pszTranslation

(OUT) Points to a buffer for the translated name.

nMaxLen

(IN) Specifies the length of the buffer for the translated name.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If *pszClassName* is NULL, default translation is provided. Otherwise, default translation is provided only if the requested specific translation is not provided.

The buffer of length *nMaxLen* should include two extra bytes for the NULL character (instead of one byte) to accomodate internationalized strings.

NCP Calls

None

See Also

NWARemovePropertyNameTranslation,
NWAAddPropertyNameTranslation

NWAGetTreeName

Gets a tree name and a browser context corresponding to an active browser window in NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAGetTreeName
   (pnstr   treeName,
    pnstr   defContext);
```

Parameters

treeName

(OUT) Points to a buffer where the tree name will be provided.

defContext

(OUT) Points to a buffer where the browser context will be provided.

Return Values

NWA_RET_SUCCESS
NWA_ERR_BROWSER_NOT_ACTIVE
NWA_ERR_INVALID_PARAMETER

Remarks

NetWare Administrator sets the tree name corresponding to an active browser as the default tree.

Snap-in must allocate the buffer specified by *treeName* to be of sufficient size to hold MAX_TREE_NAME_CHARS (defined in nwdsdefs.h) characters.

Snap-in must also allocate the buffer specified by *brwsContext* to be at least MAX_DN_BYTES in size.

NCP Calls

None

NWALaunchConfigDialog

Launches a dialog box to configure the toolbar and status bar preferences
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWALaunchConfigDialog
  (void);
```

Return Values

NWA_RET_SUCCESS
NWA_RET_ERR

Remarks

If a Snap-in MDI child window needs to provide toolbar and status bar configuration preferences, **NWALaunchConfigDialog** should be called. Calling **NWALaunchConfigDialog** ensures that the functionality will be accessable when a Snap-in MDI child window is active.

NCP Calls

None

NWALaunchDetails

Launches a details window
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWALaunchDetails
   (pnstr8   classType,
    pnstr8   className,
    pnstr8   objName);
```

Parameters

classType

(IN) Points to the class type of the object.

className

(IN) Points to the name of the class.

objName

(IN) Specifies the complete distinguished name of the object.

Remarks

NWALaunchDetails displays the details for the object specified in the preferred tree. It is assumed that administrator is authenticated to the tree. If the tree of interest is not set as the preferred tree, then Snap-in must set the preferrred tree before calling **NWALaunchDetails**.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

NCP Calls

None

NWALaunchDSFlatBrowser

Launches a DS Flat Browser

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWALaunchDSFlatBrowser
   (HWND                  hwndParent,
    nuint32               userParam,
    LPSTR                 contextStr,
    LPCSTR                selObjectStr,
    LPCSTR                navObjectStr,
    LPCSTR                selObjectLabel,
    nuint32               brwsFlags,
    NWAFlatBrowserProc    pfn);
```

Parameters

hwndParent

(IN) Specifies a parent window handle of the DS Flat Browser window.

userParam

(IN) Specifies a user parameter provided in the Flat Browser callback function.

contextStr

(IN/OUT) Specifies the DS context used by the DS Flat Browser upon input. Receives the DS context if it is changed in the Flat Browser during the call to **NWALaunchDSFlatBrowser** upon output.

selObjectStr

(IN) Specifies the string used as a caption for the edit dialog box taking the object filter input. If this is NULL, the default caption of the name filter is displayed.

navObjectStr

(IN) Specifies the string used as a caption for the edit dialog box that takes the directory context filter input. If this is NULL, the default caption “Directory Context Filter” is displayed.

selObjectLabel

(IN) Specifies the string used as a caption for the edit dialog box which displays the selected objects. If this is NULL, the default caption “Selected Object” is displayed.

brwsFlags

(IN) Specifies the configuration flags used by Flat Browser. Flags which can be ORed follow:

NWA_FB_BROWSE_PUBLIC: [PUBLIC] object can be selected.
NWA_FB_BROWSE_ROOT: [ROOT] object can be selected.
NWA_FB_SINGLE_SELECT: Allows selection of a single object.
NWA_FB_MULTIPLE_SELECT: Allows selection of multiple objects.

pfn

(IN) Specifies the Flat Browser callback function.

Remarks

NOTE: Snap-in must allocate a *contextStr* buffer of minimum size NWA_MAX_PATH_SIZE.

contextStr specifies the context to use for browsing. This context refers to the tree set as the preferred tree. If the tree of interest is different from the current preferred tree, then Snap-in must set the preferrred tree before calling **NWALaunchDSFlatBrowser**.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

NCP Calls

None

NWALaunchFSFlatBrowser

Launches an FS Flat Browser

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWALaunchFSFlatBrowser
  (HWND                 hwndParent,
   nuint32              userParam,
   LPSTR                dscontextStr,
   LPSTR                fscontextStr,
   LPCSTR               selObjectStr,
   LPCSTR               navObjectStr,
   LPCSTR               selObjectLabel,
   nuint32              brwsFlags,
   NWAFlatBrowserProc   pfn);
```

Parameters

hwndParent

(IN) Specifies a parent window handle of the FS Flat Browser window.

userParam

(IN) Specifies a user parameter provided in the Flat Browser callback function.

dscontextStr

(IN/OUT) Specifies the DS context used by the FS Flat Browser upon input. Receives the DS context if it is changed in the Flat Browser by **NWALaunchFSFlatBrowser** upon output.

fscontextStr

(IN/OUT) Specifies the FS context used by the FS Flat Browser upon input. Receives the FS context if it is changed in the Flat Browser by **NWALaunchFSFlatBrowser** upon output.

selObjectStr

(IN) Specifies the string used as a caption for the edit dialog box taking the Object Filter input. If this is NULL, the default caption of the Name Filter is displayed.

navObjectStr

(IN) Specifies the string used as a caption for the edit dialog box taking the Directory Context Filter input. If this is NULL, the default caption "Directory Context Filter" is displayed.

selObjectLabel

(IN) Specifies the string used as a caption for the edit dialog box which displays the selected objects. If this is NULL, the default caption "Selected Objects" is displayed.

brwsFlags

(IN) Specifies the configuration flags used by Flat Browser. Flags which can be ORed follow:

NWA_FB_NONDS_VOLUMES: Non DS volumes can be selected.

NWA_FB_EXPAND_VOLUMES: Volumes can be expanded to display directories and files.

NWA_FB_SINGLE_SELECT: Allows selection of a single object.

NWA_FB_MULTIPLE_SELECT: Allows selection of multiple objects.

pfn

(IN) Specifies the Flat Browser callback function.

Remarks

NOTE: Snap-in must allocate a *dscontextStr* and *fscontextStr* buffer of minimum size NWA_MAX_PATH_SIZE.

dscontextStr specifies the context to use for browsing. This context refers to the tree set as the preferred tree. If the tree of interest is different from the current preferred tree, then Snap-in must set the preferrred tree before calling **NWALaunchFSFlatBrowser**.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

NCP Calls

None

NWAMVEDProc

Is a type definition for a MVED callback proc which returns the initial values and initial values-count for the MVED control as well as the filter values for the flat browser

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK (nuint32, NWAMVEDProc)
   (nuint32   userParam,
    nuint16   uMsg,
    nparam    param1,
    nparam    param2);
```

Parameters

userParam

Specifies the user parameter provided in **NWACreateMved**.

uMsg

Specifies the message to the callback.

param1

Specifies parameter 1 of the message.

param2

Specifies parameter 2 of the message.

Messages

NWA_MSG_MVED_INITCOUNT
NWA_MSG_MVED_INITVALUE
NWA_MSG_FBFILTER_COUNT
NWA_MSG_FBFILTER_VALUE

Remarks

MvedProc is provided as a parameter to **NWACreateMved**. The callback gets invoked to return the initial values count and initial values for MVED control. **MvedProc** also gets invoked for filter values for the flat browser (if MVED values are distinguished names).

Since the MVED control is managed by MPEW window, MVED control can be used only in a Snap-in dialog page and not in any other dialog or window. **NWAGetMvedCount** and **NWAGetMvedValue** are used to get the data from MVED control.

NCP Calls

None

See Also

NWA_MSG_MVED_INITCOUNT, NWA_MSG_MVED_INITVALUE, NWA_MSG_FBFILTER_COUNT, NWA_MSG_FBFILTER_VALUE, **NWACreateMved**, **NWAGetMvedCount**, **NWAGetMvedValue**

NWAProcessToolsMenuItemCommand

Executes a command corresponding to a specified Tools menu item

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAProcessToolsMenuItemCommand
   (pnstr8            viewName,
    NWAToolsMenuItem   menuItem);
```

Parameters

viewName

(IN) Points to the name of the view, NetWare Administrator console, or MDI child window class name under which the Tools menu items are registered.

menuItem

(IN) Specifies the NWAToolsMenuItem structure which contains a Tools menu item to be processed.

Return Values

NWA_ERR_SUCCESS
NWA_ERR_INVALID_CLASSNAME
NWA_ERR_CLASSNAME_NOT_FOUND
NWA_ERR_ITEM_NOT_FOUND

Remarks

viewName should be set to NWA_VIEW_CONSOLE to execute a menu item under the Tools menu in the NetWare Administrator console.

If a Snap-in view is using its own view-specifiec menu, it is recommended that the functionality of the NetWare Administrator Tools menu items be provided under a drop-down menu in the View menu.

NWAGetToolsMenuItemCount and **NWAGetToolsMenuItem** can be called to enumerate the NetWare Administrator Tools menu items.

NWAProcessToolsMenuItemCommand can be called to execute the functionality of specific menu items under the Tools menu. Using this process, it is possible to easily simulate the Tools menu of the NetWare Administrator console window and any other Snap-in MDI child window in a Snap-in MDI child window specific menu.

NCP Calls

None

See Also

NWAGetToolsMenuItem, **NWAGetToolsMenuItemCount**, **NWAProcessToolsMenuItemValid**

NWAProcessToolsMenuItemValid

Executes a command to enable or disable a Tools menu item

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWAProcessToolsMenuItemValid
   (pnstr8              viewName,
    NWAToolsMenuItem    menuItem,
    pnuint              pFlags);
```

Parameters

viewName

(IN) Points to the name of the view, NetWare Administrator console, or MDI child window class name under which the Tools menu items are registered.

menuItem

(IN) Specifies the **NWAToolsMenuItem** function specifying the Tools menu item to be processed.

pFlags

(OUT) Points to the flags used to display the menu item.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_CLASSNAME
NWA_ERR_CLASSNAME_NOT_FOUND
NWA_ERR_ITEM_NOT_FOUND

Remarks

viewName should be set to NWA_VIEW_CONSOLE to enable/disable a menu item under the Tools menu in the NetWare Administrator console.

If a Snap-in view is using its own view-specific menu, it is recommended that the functionality of the NetWare Administrator Tools menu items be provided under drop-down menus in the View menu.

NWAGetToolsMenuItemCount and **NWAGetToolsMenuItem** can be called to enumerate the NetWare Administrator Tools menu item.

NWAProcessToolsMenuItemCommand can be called to execute the functionality of specific menu items under the Tools menu. Call **NWAProcessToolsMenuItemValid** to find out the menu flags used to display menu items; Snap-in can then used the same flags to display corresponding items in its menu.

NCP Calls

None

See Also

NWAGetToolsMenuItem, **NWAGetToolsMenuItemCount**, **NWAProcessToolsMenuItemCommand**

NWARegisterMDIChildWindow

Registers an MDI child window class name with NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARegisterMDIChildWindow
  (LPCSTR                  className,
   LPCSTR                  developerInfo,
   HINSTANCE               hDLL,
   NWASnapinMDIChildProc   classProc,
   HMENU                   hClassMenu,
   HICON                   hClassIcon,
   nuint32                 version);
```

Parameters

className

(IN) Specifies the class name of an MDI child window to be registered (non-NULL, NULL-terminated string, with a maximum length of NWA_MAX_CLASSNAMELENGTH).

developerInfo

(IN) Specifies information about the Snap-in developer (non-NULL, NULL-terminated string with a maximum length of NWA_MAX_DEVINFOLENGTH).

hDLL

(IN) Specifies the valid DLL module handle of the Snap-in DLL.

classProc

(IN) Points to the **NWASnapinMDIChildProc** MDI child window callback function (non-NULL).

hClassMenu

(IN) Specifies the menu which will be displayed on the NetWare Administrator console when the MDI child window is active (5000-10000).

hClassIcon

(IN) Specifies the icon which will be displayed when MDI child window is minimized.

version

(IN) Specifies the compile time Snap-in version number indicating the version of SNAPIN.LIB used to compile the Snap-in DLL.

Return Values

NWA_RET_SUCCESS
NWA_ERR_DUPLICATE_ENTRY
NWA_ERR_NO_MEMORY
NWA_ERR_INVALID_CALLBACK_PROC
NWA_ERR_INVALID_PARAMETER
NWA_ERR_INVALID_CLASSNAME
NWA_ERR_INVALID_MODULE_HANDLE

Remarks

NetWare Administrator will send the function pointed to by *classProc* NWA_MSG_MDICHILD_CREATED, NWA_MSG_MDICHILD_SAVESETTINGS, NWA_MSG_MDICHILD_SAVESETTINGS, and NWA_MSG_MDICHILD_RESTORE messages.

The menu Identifiers used in the *hClassMenu* parameter should be in the range of 5000 to 10000. Using this range ensures there is no menu ID conflict between NetWare Administrator and the Snap-in MDI child menu.

NetWare Administrator identifies a Snap-in MDI child window by a class name. With each such class registered, there is one callback function specified.

The *classProc* is used for saving and restoring all the MDI child windows of a specific class when NetWare Administrator shuts down and comes up again. The callback function will be a NWA_MSG_MDICHILD_CREATED message when an instance of MDI child window is created. Snap-in usually will call **NWARegisterMDIChildWindow** in **InitSnapin** implemented in a Snap-in DLL.

NCP Calls

None

See Also

InitSnapin, **NWACreateMDIChildWindow**

NWARegisterMenu

Registers menu items when they are added to the Tools menu and registers the functions to be called when the Snap-in menu item is selected

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARegisterMenu
   (pnstr8                     className,
    nuint16                    menuParentId,
    pnstr8                     menuParent,
    nuint16                    menuOption,
    pnuint16                   pmenuID,
    pnstr8                     menuString,
    pnstr8                     menuHint,
    NWASnapinMenuActionProc    menuActionProc,
    NWASnapinMenuValidProc     menuValidProc,
    nuint32                    version);
```

Parameters

className

(IN) Points to the view where the Snap-in menu item will be added. NWA_VIEW_CONSOLE and NWA_VIEW_BROWSER are the only views supported.

menuParentId

(IN) Specifies the ID of the parent menu item. The ID must be zero for the menu item to be added in the Tools menu.

menuParent

(IN) Points to the parent menu item. Must be null for the menu item to be added in the Tools menu.

menuOption

(IN) Specifies the menu options for the Snap-in menu item; must be MF_STRING.

pmenuID

(OUT) Points to the menu ID for the registered menu item.

menuString

(IN) Points to the NULL terminated string for a Snap-in menu item.

menuHint

(IN) Points to the NULL terminated string displayed as Hint on the title bar when the menu item has a focus.

menuActionProc

(IN) Specifies a user-supplied callback menu action function which is called when the menu item is selected.

menuValidProc

(IN) Specifies a user-supplied callback menu valid function which is called when the menu is pulled down.

version

(IN) Specifies the Snap-in interface version at compile time.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

NCP Calls

None

NWARegisterObjectProc

Registers an object class for the Snap-in module

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARegisterObjectProc
  (pnstr8                 classType,
   pnstr8                 className,
   pnstr8                 developerInfo,
   HINSTANCE              hDLL,
   NWASnapinObjectProc    proc,
   nuint32                version);
```

Parameters

classType

(IN) Points to the object class type. For example, FS_OBJECT_TYPE or DS_OBJECT_TYPE.

className

(IN) Points to the schema class name. Use the #defines in NWDSNMTP for existing objects.

developerInfo

(IN) Points to information about the developer.

hDLL

(IN) Specifies the Snap-in's DLL handle.

proc

(IN) Specifies the Snap-in call back function. NetWare Administrator sends messages to this function.

version

(IN) Specifies the Snap-in version at compile time.

Return Values

NWA_RET_SUCCESS

NWA_ERR_ERROR

Remarks

NWARegisterObjectProc must be called in the Snap-in's **InitSnapin**.

It is possible to register multiple procedures for the same Snap-in object class in Snap-in DLLs. DLLs are invoked in the order they are registered for the details page.

NCP Calls

None

NWARegisterObjectProcEx

Registers an object class for the Snap-in module (extended version)
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARegisterObjectProcEx
   (pnstr8                 classType,
    pnstr8                 className,
    pnstr8                 developerInfo,
    HINSTANCE              hDLL,
    NWASnapinObjectProc    proc,
    nuint32                version,
    nuint32                snapinFlags);
```

Parameters

classType

(IN) Points to the object class type. For example:

FS_OBJECT_TYPE
DS_OBJECT_TYPE

className

(IN) Points to the schema class name (use the #defines in nwdsnmtp for existing objects).

developerInfo

(IN) Points to the developer information.

hDLL

(IN) Specifies the Snap-ins DLL handle (valid module handle).

proc

(IN) Specifies the **NWASnapinObjectProc** callback function to which NetWare Administrator will send messages (non-NULL function pointer).

version

(IN) Specifies the Snap-in version at compile time.

snapinFlags

(IN) Specifies flags indicating the type of the object proc (can also specify whether the PageOption button in the MPEW windows is displayed.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_STRING_PARAM

Remarks

NWARegisterObjectProcEx achieves the same functionality as **NWARegisterObjectProc** with an additional parameter that indicates a specific scenario in which the **NWASnapinObjectProc** callback function will be called.

snapinFlags should be set to one of the following enumeration values:

NWA_SNAPIN_NORMAL	Proc to handle messages for bringing up details of an object.
NWA_SNAPIN_TEMPLATE	Proc to handle messages for bringing up details of a user template object. Currently templates are only implemented for user objects.
NWA_SNAPIN_MULTIOBJ	Proc to handle messages for editing multiple user objects at a time. Currently multiple object details are only implemented for user objects.

If Snap-in needs to register the same proc for more than one scenario, call **NWARegisterObjectProcEx** and register the proc individually for a specific scanario, as the flags in *snapinFlags* cannot be combined.

However, one of the above flags can be ORed with the NWA_SNAPIN_DISABLED_PAGESETUP flag. If the NWA_SNAPIN_DISABLED_PAGESETUP flag is set, the PageOption push button in the MPEW details window will not be displayed. If the Snap-in is only an attribute Snap-in (only adds an attribute definition to an existing object class), the NWA_SNAPIN_DISABLE_PAGESETUP flag will be ignored.

NWARegisterObjectProcEx must be called from within the Snap-ins **InitSnapin** function.

NCP Calls

None

See Also

InitSnapin, **NWARegisterObjectProc**, **NWASnapinObjectProc**

NWARegisterToolBarButton

Allows tools that Snap-in to the NetWare Administrator Tools menu to register a button to be displayed on the toolbar

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARegisterToolBarButton
  (NWAToolBarRegistrationStruct  *pStruct);
```

Parameters

pStruct

(IN) Points to the address of the NWAToolBarRegistrationStruct structure.

Return Values

NWA_RET_SUCCESS
NWA_ERR_INVALID_SNAPIN_VERSION
NWA_ERR_NO_MEMORY
NWA_ERR_INVALID_CALLBACK_PROC
NWA_ERR_INVALID_STRING_RESOURCE_ID
NWA_ERR_INVALID_STRING_PARAMETER
NWA_ERR_INVALID_BITMAP_PARAMETER
NWA_ERR_INVALID_INSTANCE_HANDLE
NWA_ERR_INVALID_MENUID
NWA_ERR_INVALID_STRUCT_SIZE
NWA_ERR_INVALID_PARAMETER
NWA_ERR_DUPLICATE_TOOLBAR_BUTTON_NAME

Remarks

The button bitmap should measure 20 x 18 pixels with 16 colors.

NCP Calls

None

NWARemoveClassData

Removes all class bitmaps from NetWare Administrator bitmap tables

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARemoveClassData
  (LPCSTR     pszClassName,
   HBITMAP   *phClassBitamp,
   HBITMAP   *phClassAliasBitmap,
   HBITMAP   *phClassReadOnlyBitmap);
```

Parameters

pszClassName

(IN) Specifies the schema class name.

phClassBitmap

(OUT) Points to the normal bitmap of the class.

phClassAliasBitmap

(OUT) Points to the alias birmap of the class.

phClassReadOnlyBitmap

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

The Snap-in calls **NWARemoveClassData** when it receives the NWA_MSG_CLOSESNAPIN message. The Snap-in should free up Windows resources.

NCP Calls

None

See Also

NWA_MSG_INITSNAPIN, NWA_MSG_CLOSESNAPIN, **NWAGetClassBitmap**

NWAGetClassReadOnlyBitmap, NWAAddClassData

NWARemovePropertyNameTranslation

Removes the translated property name for any property in NetWare Administrator

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_EXTERN_LIBRARY(NWRCODE) NWARemovePropertyNameTranslation
   (LPCSTR   pszPropertyName,
    LPCSTR   pszClassName);
```

Parameters

pszPropertyName

(IN) Specifies the schema property name.

pszClassName

(IN) Specifies the schema class name.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If *pszClassName* is NULL, the default translation for the property is removed. Passing NULL to *pszClassName* removes only the default translation, not translations specified by a class name.

NCP Calls

None

See Also

NWAGetTranslatedClassName, **NWAAddClassData**

NWASnapinMDIChildProc

Is a type definition called by NetWare Administrator to create, clear, save and restore MDI child window settings

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK (LRESULT, NWASnapinMDIChildProc)
   (pnstr8    viewName,
    nuint16   message,
    nparam    param1,
    nparam    param2);
```

Parameters

viewName

(IN) Points to the view or class name with which an MDI child window is registered.

message

(IN) Specifies the message.

param1

(IN) Specifies the message parameter one.

param2

(IN) Specifies the message parameter two.

Return Values

NWA_RET_SUCCESS

NCP Calls

None

NWASnapinMenuActionProc

Is a type definition called by NetWare Administrator when its associated Snap-in module's menu item is selected

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK (void, NWASnapinMenuActionProc)
   (void);
```

Return Values

None

Remarks

MenuAction is provided as a parameter to **NWARegisterMenu**. **MenuAction** is invoked when the Snap-in menu item is selected in NetWare Administrator.

NCP Calls

None

See Also

NWARegisterMenu

NWASnapinMenuValidProc

Is a type definition called by NetWare Administration when the user pulls down the associated Snap-in menu item

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK(void, NWASnapinMenuValidProc)
   (pnuint16   pFlags);
```

Parameters

pFlags

Points to flags which are used in displaying menu items.

Return Values

None

Remarks

MenuValidProc is provided as a parameter to **NWARegisterMenu**. **MenuValidProc** gets invoked when the Snap-in menu item is pulled down in NetWare Administrator and is used to enable/disable a menu item.

NCP Calls

None

See Also

NWARegisterMenu

NWASnapinObjectProc

Is a type definition for a Snap-in object class callback function

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK(NWRCODE, NWASnapinObjectProc)
  (pnstr8  objName,
   nuint16 message,
   nparam  param1,
   nparam  param2);
```

Parameters

objName

Points to the complete name of the object for which the callback is involved.

message

Specifies the message to the callback.

param1

Specifies parameter 1 of the message.

param2

Specifies parameter 2 of the message.

Messages

NWA_MSG_INITSNAPIN
NWA_MSG_CLOSESNAPIN
NWA_MSG_CREATEOBJECT
NWA_MSG_GETPAGECOUNT
NWA_MSG_REGISTERPAGE
NWA_MSG_RENAME
NWA_MSG_MODIFY
NWA_MSG_QUERYDELETE
NWA_MSG_QUERYCOPY
NWA_MSG_QUERYMOVE
NWA_MSG_GETVALIDOPERATIONS

Remarks

ObjectclassProc is provided as a parameter to **NWARegisterObjectProc**. **ObjectclassProc** is called by NetWare Administrator with messages to initialize or close the Snap-in object class, register the details pages, get valid oeprations, etc.

Make sure the callback function is FAR PASCAL and exported out of the DLL.

NCP Calls

None

See Also

NWA_MSG_INITSNAPIN, NWA_MSG_CLOSESNAPIN, NWA_MSG_CREATEOBJECT, NWA_MSG_GETPAGECOUNT, NWA_MSG_REGISTERPAGE, NWA_MSG_RENAME, NWA_MSG_MODIFY, NWA_MSG_QUERYDELETE, NWA_MSG_QUERYCOPY, NWA_MSG_QUERYMOVE, NWA_MSG_GETVALIDOPERATIONS, **NWARegisterObjectProc**

NWASnapinTBButtonEnableProc

Is a type definition called by NetWare Administration to find out if a toolbar button registered by a tools menu Snap-in item should be enabled or disabled

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

N_TYPEDEF_CALLBACK(void,NWASnapinTBButtonEnableProc)
   (nint      menuID,
    pnbool    pfEnabled);
```

Parameters

menuID

(IN) Specifies the button menu ID to enable or disable.

pfEnabled

(OUT) Points to N_TRUE if the button should be enabled and N_FALSE if the button should be disabled.

Return Values

N_TRUE
N_TRUE

NCP Calls

None

PostInitSnapin

Is sent by NetWare Administrator after all Snap-in DLLs have been loaded and initialized

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

void FAR PASCAL PostInitSnapin
  (HWND    hwndAdmin);
```

Parameters

hwndAdmin

(IN) Specifies the window handle of the NetWare Administrator frame window.

Return Values

None

Remarks

PostInitSnapin is implemented in a Snap-in DLL and is called by NetWare Administrator after NetWare Administrator has loaded and initialized all the Snap-in DLLs.

Snap-in does not have to implement **PostInitSnapin**.

NetWare Administrator will only call **PostInitSnapin** if it is available.

NCP Calls

None

ShutDown

Is called by NetWare Administrator when Snap-in modules are being closed

Syntax

```
#include <nwsnapin.h>

void PAR PASCAL ShutDown
  (void);
```

Return Values

None

Remarks

ShutDown must provide all Snap-in DLL specific cleanup.

See Also

InitSnapin

Snap-in: Structures

NWACreateMDIChildParam

Provides the parameters used to create an MDI child window

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct tagNWACreateMDIChildParam
{
   nuint32            structSize;
   WINDOWPLACEMENT    wndpl;
} NWACreateMDIChildParam;
```

Fields

structSize

Specifies the size of the NWACreateMDIChildParam structure.

wndpl

Specifies the window position parameters which are used to create an MDI child window.

Remarks

WINDOWPLACEMENT is a Microsoft Windows* defined structure and can be found in the Microsoft Windows SDK documentation.

NWAPageStruct

Defines configuration information for the pages in MPEW of NWAdmin

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct
{
   DLGPROC     dlgProc;
   HINSTANCE   hDLL;
   LPSTR       resName;
   LPSTR       pageTitle;
   LPARAM      initParam;
} NWAPageStruct;
```

Fields

digProc

Specifies the dialog procedure.

hDLL

Specifies the resource DLL handle.

resName

Points to the resource name.

pageTitle

Points to the title of the page in the listbox.

initParam

Specifies the initialization parameters.

NWASelObject, PNWASelObject

Defines configuration information for the type of object selected from the NWADMIN View

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct
{
   char   objType[NWA_MAX_CLASS_TYPE];
   char   objClass[NWA_MAX_CLASS_NAME];
   char   objName[NWA_MAX_OBJECT_NAME];
} NWASelObject, N_FAR *PNWASelObject;
```

Fields

objType

Specifies “DS”, “FS”, or some user defined object type.

objClass

Specifies the object class name. If *objType* specifies “FS”, then *objClass* specifies a File and Directory object class name. If *objType* is “DS”, then *objClass* specifies a DS object class name.

objName

Specifies the complete object class name or path.

NWAStatusBarItemStruct

Provides the information necessary to add text fields to the status of the toolbar

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct
{
   nuint32   lStructSize;
   nint      itemId;
   nuint     maxChars;
   pnstr8    pszText;
} NWAStatusBarItemStruct;
```

Fields

lStructSize

Specifies the length (in bytes) of the NWAStatusBarItemStruct structure.

itemId

Specifies the unique identifier of the item in the status bar.

maxChars

Specifies the maximum number of characters that can be displayed in the text field.

pszText

Points to the text that will be displayed in the text field when the text field is created.

NWAStatusBarPrefItemStruct

Provides the necessary information to add an item to the status bar preferences dialog box

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct
{
   nuint32     lStructSize;
   nint        itemId;
   HINSTANCE   hResourceDLL;
   nint        textResId;
   nint        hintResId;
   nuint       maxChars;
} NWAStatusBarPrefItemStruct;
```

Fields

lStructSize

Specifies the length (in bytes) of the NWAStatusBarPrefItemStruct structure.

itemId

Specifies the unique identifier of the field in the status bar.

hResourceDLL

Specifies the instance handle of the DLL containing the string resources corresponding to the *textId* and *hintId* structure fields.

textResId

Specifies the resource identifier of the string describing the status bar option.

hintResId

Specifies the resource identifier of the string describing the nature of the status bar option.

maxChars

Specifies the maximum number of characters that can be displayed in the text field.

Remarks

The text specified by the *textResId* structure field will appear in the status bar preferenced dialog listbox.

The text specified by the *hintResId* structure field will appear in the status bar preferences dialog box when the mouse is over the corresponding status bar dialog listbox item.

The number specified by the *maxChars* structure field should be the same value as in the corresponding NWAStatusBarItemStruct structure.

NWAToolBarItemStruct

Provides the necessary information to add buttons and spaces to the toolbar
Service: Snap-in
Defined In: nwsnapin.h

Structure

```
typedef struct
{
   nuint32     lStructSize;
   HINSTANCE   hResourceDLL;
   nint        menuId;
   nint        hintId;
   nint        bitmapId;
   nint        toolTipStrId;
   TBState     buttonState;
   TBType      buttonType;
   nbool       fEnabled
} NWAStatusBarPrefItemStruct;
```

Fields

lStructSize

Specifies the length (in bytes) of the NWAStatusBarPrefItemStruct structure.

hResourceDLL

Specifies the handle to the resource DLL containing the bitmap resource corresponding to the *bitmapId* structure field.

menuId

Specifies the unique identifier of the button in the toolbar.

hintId

Specifies the resource ID of the string to be displayed in the NWAdmin caption when the user's mouse flied over the corresponding toolbar button (if set to zero, no hint will be displayed).

bitmapId

Specifies the resource ID of a bitmap resource to appear on the button.

toolTipStrId

Specifies the resource ID of the string to be displayed in the tool tip (if set to zero, no tool tip will be displayed).

buttonState

Specifies the default state of the button.

buttonType

Specifies the type of the button.

fEnabled

Specifies whether the button is enabled when it is first created.

Remarks

When a view wants to add an item to the toolbar, it sends a NWA_MSG_TOOLBAR_ADDITEM message to the NWAdmin main window.

The *lParam* parameter for each desired item. parameter of the NWA_MSG_TOOLBAR_ADDITEM message contains a long pointer to a NWAToolBarItemStruct structure.

If the button specified by the *menuId* structure field has a *buttonType* of Command, the *menuId* structure field should correspond to the command ID for the menu item it represents.

The enumerated type, TBState, corresponding to the *buttonState* structure field can be TBUp, TBDown, or TBIndeterminate. The indetereminate state is an intermediate state between down and up.

The enumerated type, TBType, has three possible values described below:

TBCommand	Specifies the button is a command.
TBToggle	Specifies the button is a toggle button.

Setting buttons may be used to indicate the state of something, similar to toolbar buttons in WordPerfect for Windows that shows if selected text is bold, italicized, or underlined.

NWAToolBarPrefItemStruct

Provides the necessary information to add an item to the toolbar preferences dialog box

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct
{
   nuint32     lStructSize;
   nint        menuId;
   HINSTANCE   hResourceDLL;
   nint        itemStrId;
   nint        hintStrId;
   nint        bitmapId;
} NWAToolBarPrefItemStruct;
```

Fields

lStructSize

Specifies the length (in bytes) of the NWAToolBarPrefItemStruct structure.

menuId

Specifies the unique identifier of the button in the toolbar.

hResourceDLL

Specifies the handle to the resource DLL containing the bitmap resource corresponding to the *bitmapId*, *itemStrId*, and *hintStrId* structure fields.

itemStrId

Specifies the resource ID of the string to be displayed in the listbox of available items.

hintStrId

Specifies the resource ID of the string describing the toolbar item to be displayed in the hint static field when the corresponding item is selected in the listbox of available items.

bitmapId

Specifies the resource ID of a bitmap resource to appear on the button.

Remarks

The toolbar preferences dialog box allows users to specify which items they want to be visible in the toolbar.

they want to be visible in the toolbar.

When a view receives a NWA_MSG_TOOLBAR_QUERYITEMCOUNT message, the view returns the number of items it wants to add to the toolbar preferences dialog box. NWAdmin will then send a NWA_MSG_TOOLBAR_ADDPREFITEM message containing a pointer to a NWAToolBarPrefItemStruct structure in the *lParam* parameter for each desired item.

If the button specified by the *menuId* structure field has a *buttonType* of TBCommand, the *menuId* structure field should correspond to the command ID for the menu item it represents.

NWAToolBarRegistrationStruct

Provides the necessary information for a Snap-in item under the NWAdmin Tools menu to register an item for the toolbar and the toolbar preferences dialog box

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct
{
   nuint32                        lStructSize;
   pnstr8                         nameOfSnapinButton;
   nint                           menuId;
   HINSTANCE                      hResourceDLL;
   nint                           bitmapId;
   nint                           textStrId;
   nint                           hintStrId;
   nint                           toolTipStrId;
   TBState                        buttonState;
   TBType                         buttonType;
   NWASnapinTBButtonEnableProc    enableProc;
   nuint32                        version;
} NWAToolBarRegistrationStruct;
```

Fields

lStructSize

Specifies the length (in bytes) of the NWAToolBarRegistrationStruct structure.

nameOfSnapinButton

Points to the unique string identifier of the button in the toolbar (maximum of NWA_MAX_SNAPIN_BUTTON_NAME).

menuId

Specifies the menu ID returned by the **NWARegisterMenu** function.

hResourceDLL

Specifies the handle to the resource DLL containing the bitmap resource corresponding to the *bitmapId*, *textStrId*, *hintStrId*, and *toolTipStrId* structure fields.

bitmapId

Specifies the resource ID of the bitmap resource to appear on the button.

textStrId

Specifies the resource ID of the string to be displayed in the listbox of available items in the toolbar preferences dialog box.

hintStrId

Specifies the resource ID of the descriptive string to be displayed in the hint static field when the corresponding item is selected in the listbox of available items.

toolTipStrId

Specifies the resource ID of the string to be displayed in the tool tip for the button.

buttonState

Specifies the default state of the button.

buttonType

Specifies the type of the button.

enableProc

Points to the **NWASnapinTBButtonEnableProc** callback function used to determine if the button should be enabled or disabled.

version

Specifies the Snap-in version at compile time.

Remarks

A pointer to the NWAToolBarRegistrationStruct structure is passed as a parameter to the **NWARegisterToolBarButton** function.

The string pointed to by the *nameOfSnapinButton* structure field should contain the Snap-in writer's company name and application name as well as additional information to distinguish multiple buttons from each other.

The enumerated type, TBState, can be set to: TBUp, TBDown, or TBIndeterminate. The indeterminate state is an intermediate state between down and up.

The enumerated type, TBType, has three possible values described below:

TBCommand	Specifies the button is a command.
TBToggle	Specifies the button is a toggle button.

Setting buttons may be used to indicate the state of something, similar to the toolbar buttons in WordPerfect for Windows that indicate if selected text is bold, italicized, or underlined.

See Also

NWARegisterMenu, NWASnapinTBButtonEnableProc

NWAToolsMenuItem

Contains information about a menu item under the Tools drop-down menu in an MDI child window

Service: Snap-in

Defined In: nwsnapin.h

Structure

```
typedef struct tagNWAToolsMenuItem
{
   nuint32    structSize;
   nuint      parentMenuId;
   nuint      menuId;
   nuint      menuOption;
   char       szMenuText[32];
   char       szMenuHintText[128];
} NWAToolsMenuItem;
```

Fields

structSize

Specifies the size of the NWAToolsMenuItem structure.

parentMenuId

Specifies the ID of the parent menu item.

menuId

Specifies the ID of the menu item.

menuOption

Specifies the menu item options used to create the menu item (MF_STRING, MF_POPUP, etc.)

szMenuText

Points to the menu item text string.

szMenuHintText

Points to the menu item hint text string.

See Also

NWAGetToolsMenuItemCount

Snap-in: Messages

NWA_MSG_APPLYTEMPLATE

Is sent to the Snap-in(s) with the object name being created and the template object name being used to create the object

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies NWA_MSG_APPLYTEMPLATE.

p1

Specifies the template object name being used to create the object.

p2

Is currently not being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

Snap-in should process the NWA_MSG_APPLYTEMPLATE message by applying the template object attribute values to the attribute values of the object being created.

NWA_MSG_CLOSESNAPIN

Is the last message a Snap-in receives and it should unregister its translated names and bitmaps, and free allocated memory

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies NWA_MSG_CLOSESNAPIN.

p1 and *p2*

Are currently not being used.

Return Values

NWA_RET_SUCCESS

See Also

NWA_MSG_INITSNAPIN

NWA_MSG_CREATEOBJECT

Is sent by NetWare Administrator to the Snap-in when a user selects the object class supported by the Snap-in from the Create dialog in the Object menu

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the parent container name where the new object must be created.

msg

Specifies NWA_MSG_CREATEOBJECT.

p1 and *p2*

Are currently not being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR
NWA_RET_SHOWDETAILS
NWA_RET_CREATEANOTHER

Remarks

If NWA_ERR_ERROR is returned, the Snap-in should show a message dialog box detailing the error.

A Snap-in should show a Create dialog box in response to the NWA_MSG_CREATOBJECT message. This dialog box should prompt the user for necessary properties to create a new object in the NetWare Directory. The user interface for the Snap-in should be similar to the create dialog used by NetWare Administrator.

NWA_MSG_FBFILTER_COUNT

Is the first message a Flat Browser or MVED callback function receives

Syntax

```
#include <nwsnapin.h>

FlatBrowserProc(nuint32 userParam, nuint16 msg, nparam p1, nparam p2);
```

Parameters

userParam

Specifies the user parameter.

msg

Specifies NWA_MSG_FBFILTER_COUNT.

p1

Specifies the count of class filters.

p2

Is not currently being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

The callback function will receive the NWA_MSG_FBFILTER_VALUE message as many times as indicated by the count.

See Also

NWA_MSG_FBFILTER_VALUE, NWA_MSG_FBOBJECT_COUNT, NWA_MSG_FBOBJECT_VALUE

NWA_MSG_FBFILTER_VALUE

Is sent to the Flat Browser or MVED callback function with the filter index of the *p1* parameter

Syntax

```
#include <nwsnapin.h>

FlatBrowserProc(nuint32 userParam, nuint16 msg, nparam p1, nparam p2);
```

Parameters

userParam

Specifies the user parameter.

msg

Specifies NWA_MSG_FBFILTER_VALUE.

p1

Is not currently being used.

p2

Specifies the buffer containing the filter value.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

See Also

NWA_MSG_FBFILTER_COUNT, NWA_MSG_FBOBJECT_COUNT, NWA_MSG_FBOBJECT_VALUE

NWA_MSG_FBOBJECT_COUNT

Is sent to the Flat Browser callback function with the count of selected objects in the flat browser, if the OK button is pressed

Syntax

```
#include <nwsnapin.h>

FlatBrowserProc(nuint32 userParam, nuint16 msg, nparam p1, nparam p2);
```

Parameters

userParam

Specifies the user parameter.

msg

Specifies NWA_MSG_FBOBJECT_COUNT.

p1 and *p2*

Are not currently being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

See Also

NWA_MSG_FBFILTER_COUNT, NWA_MSG_FBFILTER_VALUE, NWA_MSG_FBOBJECT_VALUE

NWA_MSG_FBOBJECT_VALUE

Is sent to the Flat Browser callback function with the selected object index

Syntax

```
#include <nwsnapin.h>

FlatBrowserProc(nuint32 userParam, nuint16 msg, nparam p1, nparam p2);
```

Parameters

userParam

Specifies the user parameter.

msg

Specifies NWA_MSG_FBOBJECT_VALUE.

p1

Specifies the index of the object selected.

p2

Specifies the NWASelObject, PNWASelObject structure indicating the object selected.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

The NWASelObject, PNWASelObject structure defines configuration information for the type of object selected from the NetWare Administrator view.

See Also

NWA_MSG_FBFILTER_COUNT, NWA_MSG_FBFILTER_VALUE, NWA_MSG_FBOBJECT_COUNT

NWA_MSG_GETPAGECOUNT

Is sent by NetWare Administrator to the Snap-in when a user selects the object supported by the Snap-in and then selects Details from the Object menu

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies NWA_MSG_GETPAGECOUNT.

p1 and *p2*

Are not currently used.

Return Values

NWA_MSG_GETPAGECOUNT returns the number of pages the Snap-in wants to put in the Multi-Page Edit Window (MPEW).

See Also

NWA_MSG_REGISTERPAGE

NWA_MSG_GETVALIDOPERATIONS

Is sent by NetWare Administrator to **SnapinObjectProc** function when a Snap-in object is selected and the user selects a menu operation

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object whose properties have been modified.

msg

Specifies NWA_MSG_GETVALIDOPERATIONS.

p1 and *p2*

Are not currently being used.

Return Values

NWA_OP_CONTAINER
NWA_OP_CREATE
NWA_OP_DELETE
NWA_OP_DETAILS
NWA_OP_DSTYPE
NWA_OP_FSTYPE
NWA_OP_MOVE
NWA_OP_RENAME
NWA_OP_SEARCH
NWA_OP_USERTEMPLATE

Remarks

The return value is the sum of the defined constants for which there are valid operations.

NWA_OP_CONTAINER specifies whether an object is a container and enables the Create menu item.

NWA_OP_CREATE adds a Snap-in object class name to the class list in the New Object dialog.

NWA_OP_DELETE enables Delete menu item.

NWA_OP_DETAILS enables Details menu item.

NWA_OP_DSTYPE enables Rights to other objects and Trustees of the

object menu item.
NWA_OP_FSTYPE enables Move and Copy menu items.
NWA_OP_MOVE enables Move menu item.
NWA_OP_RENAME enables Rename menu item.
NWA_OP_SEARCH enables Search menu item.
NWA_OP_USERTAMPLATE enables User Template menu item.

NWA_MSG_INITSNAPIN

Is the first message a Snap-in receives indicating it should register bitmaps, translated property, and class members

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies NWA_MSG_INITSNAPIN.

p1 and *p2*

Are currently not being used

Return Values

NWA_MSG_SUCCESS
NWA_ERR_ERROR

Remarks

If the Snap-in returns NWA_ERR_ERROR, NetWare Administrator will terminate communication with the Snap-in.

See Also

NWA_MSG_CLOSESNAPIN

NWA_MSG_IVEGOTFOCUS

Is sent by an MDI child window to the NetWare Administrator main window when the child window receives a WM_SETFOCUS message

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = (WPARAM)hwndMDIChild;
lParam = 0;
```

Parameters

wParam

Specifies the handle of the MDI child window.

lParam

Is currently not used; must be zero.

Remarks

NetWare Administrator responds to the NWA_MSG_IVEGOTFOCUS message by sending the MDI child window the NWA_MSG_TOOLBAR_POPULATE and NWA_MSG_STATUSBAR_POPULATE messages.

Each MDI child window type must have a registered Windows class name if it wants to take control of the toolbar. If the MDI child window does not want to have control of the toolbar, that child window will receive a NWA_MSG_TOOLBAR_POPULATE or NWA_MSG_STATUSBAR_POPULATE message.

NWA_MSG_MDICHILD_CLEARSETTINGS

Is sent to the **NWASnapinMDIChildProc** callback proc by NetWare Administrator when NetWare Administrator is going down and the Save settings option in NetWare Administrator is set

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

MDIChildCallbackProc (char* classname, UINT msg, nparam p1, nparam p2);
```

Parameters

classname

Points to the name of the Snap-in view class.

msg

Specifies NWA_MSG_MDICHILD_CLEARSETTINGS.

p1

Specifies the MDI child window going down with NetWare Administrator.

p2

Specifies the integral order of the MDI child window.

Return Values

NWA_RET_SUCCESS
NWA_RET_ERROR

Remarks

The NetWare Administrator browser view saves the setting the browser window position and browser context in nwadmin.ini. Snap-ins should similarly manage their own information.

One of the important pieces of information that are displayed about the view (the MDI child window) is the position of the window on the screen (window placement). Snap-in should save the window placement and restore the MDI child window to the same window placement in NetWare Administrator.

NWA_MSG_MDICHILD_CREATED

Is sent to the **NWASnapinMDIChildProc** callback proc by NetWare Administrator when an instance of the MDI child window is created

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

MDIChildCallbackProc (char* classname, UINT msg, nparam p1, nparam p2);
```

Parameters

classname

Points to the name of the Snap-in view class.

msg

Specifies NWA_MSG_MDICHILD_CREATED.

p1

Specifies the parent window handle of the MDI child window created.

p2

Specifies a window handle that must be filled by the Snap-in with a window handle which will be used by NetWare Administrator to send MDI child window related messages to the Snap-in.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

NWA_MSG_MDICHILD_RESTORE

Is sent to the **NWASnapinMDIChildProc** callback proc by NetWare Administrator when NetWare Administrator comes up

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

MDIChildCallbackProc (char* classname, UINT msg, nparam p1, nparam p2);
```

Parameters

classname

Points to the name of the Snap-in view class.

msg

Specifies NWA_MSG_MDICHILD_RESTORE.

p1 and *p2*

Are currently not being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

NWA_MSG_MDICHILD_SAVESETTINGS

Is sent to the **NWASnapinMDIChildProc** callback proc by NetWare Administrator when NetWare Administrator is going down and the Save settings option in NetWare Administrator is set

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

MDIChildCallbackProc (char* classname, UINT msg, nparam p1, nparam p2);
```

Parameters

classname

Points to the name of the Snap-in view class.

msg

Specifies NWA_MSG_MDICHILD_SAVESETTINGS

p1

Specifies the MDI child window handle.

p2

Specifies the integral order of the MDI child window going down.

Return Values

NWA_RET_SUCCESS
NWA_RET_ERROR

Remarks

The NetWare Administrator browser view saves the setting the browser window position and browser context in nwadmin.ini. Snap-ins should similarly manage their own information.

One of the important pieces of information that are displayed about the view (the MDI child window) is the position of the window on the screen (window placement). Snap-in should save the window placement and restore the MDI child window to the same window placement in NetWare Administrator.

NWA_MSG_MODIFY

Is sent by NetWare Administrator to the Snap-in when the user selects the OK button from the MPEW after making modifications

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object whose properties have been modified.

msg

Specifies NWA_MSG_MODIFY.

p1 and *p2*

Are currently not being used.

Return Values

NWA_MSG_SUCCESS
NWA_MSG_ERROR

Remarks

The Snap-in can save its page information page by page in the NWA_WM_CANCLOSE message of the dialog or it can save all of its page information at once with NWA_MSG_MODIFY.

If NWA_ERR_ERROR is returned, the Snap-in should display the appropriate error message. NetWare Administrator will not close the MPEW, allowing the user to rectify their error.

NWA_MSG_MPEWCLOSE

Is sent by NetWare Administrator to the Snap-in when the details window (MPEW) is being closed

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies

p1

Specifies the ID of the button pressed in the details window (IDOK or IDCANCEL).

p2

Is currently not being used.

Return Values

NWA_RET_SUCCESS

NWA_MSG_MULTIOBJ_COUNT

Is sent to the **NWASnapinObjectProc** callback proc indicating the number of objects selected in the browser to perform multiple object details

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies NWA_MSG_MULTIOBJ_COUNT.

p1

Specifies the number of objects selected in the browser for multiple object details.

p2

Is currently not being used.

Return Values

NWA_RET_SUCCESS

Remarks

Currently, Multiple Object Details is only implemented for user objects.

NWSnapinObjectProc will be sent a NWA_MSG_MULTIOBJ_NAME message the number of times indicated by *objCount* with a complete DN name of the selected object from NWA_MSG_MULTIOBJ_NAME.

NWA_MSG_MULTIOBJ_NAME

Is sent to the **NWASnapinObjectProc** callback proc with the complete DN name of the selected object

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the object name.

msg

Specifies NWA_MSG_MULTIOBJ_NAME.

p1

Specifies the index of the object being selected.

p2

Specifies the complete DN name for the object selected for multiple object details.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

The Snap-in should modify the object details in accordance with the NWA_MSG_MULTIOBJ_NAME message.

See Also

NWA_MSG_MULTIOBJ_COUNT

NWA_MSG_MVED_INITCOUNT

Is the first message an MVED callback function receives indicating the number of MVED values to initialize

Syntax

```
#include <nwsnapin.h>

MvedProc(nuint32 userParam, nuint16 uMsg, nparam p1, nparam p2);
```

Parameters

userParam

Specifies the user parameter.

uMsg

Specifies NWA_MSG_MVED_INITCOUNT.

p1

Specifies the count of values in MVED.

p2

Is not currently being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

The **MvedProc** callback function will receive the MWA_MSG_MVED_INITVALUE as many times as indicated by the count.

See Also

NWA_MSG_MVED_INITVALUE

NWA_MSG_MVED_INITVALUE

Is sent to the MVED callback function with an index of the MVED value indicated by the *p1* parameter

Syntax

```
#include <nwsnapin.h>

MvedProc(nuint32 userParam, nuint16 uMsg, nparam p1, nparam p2);
```

Parameters

userParam

Specifies the user parameter.

uMsg

Specifies NWA_MSG_MVED_INITVALUE.

p1

Is not currently being used.

p2

Specifies a buffer containing the MVED value.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

See Also

NWA_MSG_MVED_INITCOUNT

NWA_MSG_NOTIFYCREATEOBJECT

Is sent by NetWare Administrator once the object is created

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, nuint16 msg, nparam p1, nparam p2);
```

Parameters

name

Points to the selected copied object.

msg

Specifies NWA_MSG_NOTIFYCREATEOBJECT.

p1 and *p2*

Are currently not used.

Return Values

NWA_RET_SUCCESS

NWA_MSG_NOTIFYDELETE

NetWare Administrator sends this notification message to Snap-in when the object is deleted

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, nuint16 msg, nparam p1, nparam p2);
```

Parameters

name

Points to the selected copied object.

msg

Specifies NWA_MSG_NOTIFYDELETE.

p1 and *p2*

Are not currently used.

Return Values

NWA_RET_SUCCESS

NWA_MSG_NOTIFYMOVE

NetWare Administrator sends this notification message to Snap-in when the object is moved

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, nuint16 msg, nparam p1, nparam p2);
```

Parameters

name

Points to the selected copied object.

msg

Specifies NWA_MSG_NOTIFYMOVE.

p1

Specifies the source context string from where the object is moved.

p2

Specifies the destination context string to where the object is moved.

Return Values

NWA_RET_SUCCESS

NWA_MSG_NOTIFYRENAME

NetWare Administrator sends this notification message to Snap-in when the object is renamed

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, nuint16 msg, nparam p1, nparam p2);
```

Parameters

name

Points to the selected copied object.

msg

Specifies NWA_MSG_NOTIFYRENAME.

p1

Specifies the complete distinguished old object name before it is renamed.

p2

Is not currently being used.

Return Values

NWA_RET_SUCCESS

NWA_MSG_QUERYCOPY

Is sent by NetWare Administrator to the Snap-in when the user selects an object that is supported by the Snap-in and then selects Copy from the Object menu

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object to be copied.

msg

Specifies NWA_MSG_QUERYCOPY.

p1 and *p2*

Are not currently being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If the return value is NWA_RET_SUCCESS, NetWare Administrator will copy the object.

If the return value is NWA_ERR_ERROR, NetWare Administrator will not copy the object. A message dialog box should tell the user the reason why the object cannot be copied.

NWA_MSG_QUERYDELETE

Is sent by NetWare Administrator to the Snap-in when the user selects an object that is supported by the Snap-in and then selects Delete from the Object menu

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object to be deleted.

msg

Specifies NWA_MSG_QUERYDELETE.

p1 and *p2*

Are not currently being used.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If the return value is NWA_ERR_ERROR, NetWare Administrator will not delete the object. If NWA_RET_SUCCESS is returned, NetWare Administrator will delete the object.

If the Snap-in returns NWA_ERR_ERROR, a message dialog box should tell the user the reason why the object cannot be deleted.

NWA_MSG_QUERYMOVE

Is sent by NetWare Administrator to the Snap-in when the user selects an object that is supported by the Snap-in and then selects Move from the Object menu

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object to be moved.

msg

Specifies NWA_MSG_QUERYMOVE.

p1

Specifies the destination container where the object is being moved.

p2

Is currently not being used.

Return Values

NWA_ERR_ERROR
NWA_RET_SUCCESS

NWA_MSG_REGISTERPAGE

Is sent by NetWare Administrator to the Snap-in "n" times, where "n" is the number returned by the NWA_MSG_GETPAGECOUNT message

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object.

msg

Specifies NWA_MSG_REGISTERPAGE.

p1

Specifies the number of the page to be registered.

p2

Specifies the NWAPageStruct structure which the Snap-in fills with its page information.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

See Also

NWAPageStruct, NWA_MSG_GETPAGECOUNT

NWA_MSG_RENAME

Is sent by NetWare Administrator to the Snap-in when a user selects the object supported by the Snap-in and then selects Rename from the Object menu

Syntax

```
#include <nwsnapin.h>

SnapinObjectProc(char* name, UINT msg, unsigned long p1, unsigned long
```

Parameters

name

Points to the selected object.

msg

Specifies NWA_MSG_RENAME.

p1

Specifies a buffer to hold the new name.

p2

Is currently not being used.

Return Values

NWA_ERR_ERROR
NWA_RET_SUCCESS
NWA_RET_DODEFAULT

Remarks

A Snap-in can use its own rename dialog and complete the rename operation, or it can request NetWare Administrator to use its default rename operation.

After renaming the object, the Snap-in should return NWA_RET_SUCCESS. The Snap-in should copy the new name into the buffer pointed to by the p1.

To use the NetWare Administrator renaming method, the Snap-in should return NWA_RET_DODEFAULT.

NWA_MSG_STATUSBAR_ADDITEM

Is sent to the NetWare Administrator main window after it sends a NWA_MSG_STATUSBAR_POPULATE message to the top MDI child window

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = 0;
lParam = (LPARAM)pStatusBarItem;
```

Parameters

wParam

Is currently not being used; must be zero.

lParam

Points to the NWAStatusBarItemStruct structure containing the data required to add an item to the status bar.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

If the item was successfully added to the status bar, NWA_RET_SUCCESS will be returned by the NWA_MSG_STATUSBAR_ADDITEM message.

The items will appear on the status bar in the order they are added. The items should be added to the status bar in the order indicated by the values returned by the NWA_MSG_STATUSBAR_NEWACTIVEITEMS message.

If an item is added whose *itemId* matches a previous items *itemId*, the previous item will be removed from the status bar. In general, the last item to be added to the status bar with a duplicate *itemId* will be added to the status bar while the previous item with a matching *itemId* will be deleted from the status bar.

See Also

NWAStatusBarItemStruct

NWA_MSG_STATUSBAR_ADDPREFITEM

Is sent to the MDI child window by the NetWare Administrator main window according to the number returned by the NWA_MSG_STATUSBAR_QUERYITEMCOUNT message

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nPrefCnt = (nuint16)wParam;
ptbPrefItem = (NWAStatusBarPrefItemStruct *)lParam;
```

Parameters

nPrefCnt

Specifies the sequence number of the preference items.

ptbPrefItem

Points to the NWAStatusBarPrefItemStruct structure containing the data necessary to display an item in the status bar preferences dialog box.

Remarks

The items will appear in the status bar preference dialog box in the order they are added.

The MDI child window fills in the NWAStatusBarPrefItemStruct structure.

The results will be displayed in the status bar preferences dialog box which allows you to select which status bar items will be visible in the status bar.

NWA_MSG_STATUSBAR_DBLCLK

Is sent to an MDI child window when the user double-clicks the left mouse button over a field in the status bar

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

itemId = (nint)wParam;
pszText = (pnstr8)lParam;
```

Parameters

itemId

Specifies the unique identifier of the field in the status bar.

pszText

Points to the text contained in the status bar field indicated by the *itemId* parameter.

Remarks

The NWA_MSG_STATUSBAR_DBLCLK message gives the MDI child window the opportunity to launch a dialog box or take another action depending on which field was selected.

The *itemId* parameter corresponds to the *itemId* structure field in the NWAStatusBarItemStruct structure.

If the status bar field is empty, the *pszText* parameter will be NULL.

NWA_MSG_STATUSBAR_NEWACTIVEITEMS

Is sent to a view by the NetWare Administrator main window when a user has pressed the OK button in the status bar preferences dialog box

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nCount = (nuint16)wParam;
pnArray = (pnint)lParam;
```

Parameters

nCount

Specifies the number of active items selected by the user.

pnArray

Points to an array of *nCount* integers. If the *nCount* parameter is set to zero (0), *pnArray* will be NULL.

Remarks

The NWA_MSG_STATUSBAR_NEWACTIVEITEMS message indicates the user's status bar preferences have changed.

Each integer value in the array pointed to by *pnArray* is the unique identifier of an item that should be active in the status bar. The order of the items in the array indicates the preferred order of items in the views portion of the status bar.

Before the NetWare Administrator main window sends a NWA_MSG_STATUSBAR_NEWACTIVEITEMS message to the view, it will depopulate the status bar. After the NetWare Administrator main window handles the NWA_MSG_STATUSBAR_NEWACTIVEITEMS message, the view will receive a NWA_MSG_STATUSBAR_POPULATE message.

The view is responsible for saving and restoring its own preferences.

See Also

NWA_MSG_STATUSBAR_QUERYACTIVECOUNT, NWA_MSG_STATUSBAR_QUERYACTIVEITEMS

NWA_MSG_STATUSBAR_POPULATE

Is sent to the top MDI child window after the NetWare Administrator main window has handled a NWA_MSG_IVEGOTFOCUS message

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

hwndMain = (HWND)wParam;
```

Parameters

hwndMain

Specifies the window handle which identifies the NetWare Administrator main window that received the NWA_MSG_IVEGOTFOCUS message and sent the NWA_MSG_STATUSBAR_POPULATE message (value of *wParam*).

Remarks

When a window receives the NWA_MSG_STATUSBAR_POPULATE message, it can respond by sending a NWA_MSG_STATUSBAR_ADDITEM message to the *hwndMain* parameter.

See Also

NWA_MSG_TOOLBAR_POPULATE

NWA_MSG_STATUSBAR_QUERYACTIVECOUNT

Is sent to an MDI child window when the user opens the status bar preferences dialog box

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>
```

Return Values

The view should return the number of active items in the status bar.

Remarks

The view should return the number of active items in the status bar.

The MDI child window responds to the NWA_MSG_STATUSBAR_QUERYACTIVECOUNT message by returning the number of items that will be in the active item list in the status bar preferences dialog box.

The dialog box allows the user to determine which items will actually be visible in the status bar and each items placement within the status bar.

If the MDI child window returns a value greater than zero, the NetWare Administrator main window will respond by sending the NWA_MSG_STATUSBAR_QUERYACTIVEITEMS message. The NWA_MSG_STATUSBAR_QUERYACTIVEITEMS message will provide the address of an array of integers whose size corresponds to the value returned by the NWA_MSG_STATUSBAR_QUERYACTIVECOUNT message.

See Also

NWA_MSG_STATUSBAR_NEWACTIVEITEMS, NWA_MSG_STATUSBAR_QUERYACTIVEITEMS

NWA_MSG_STATUSBAR_QUERYACTIVEITEMS

Is sent to an MDI child window when the user opens the status bar preferences dialog box and has responded to a NWA_MSG_STATUSBAR_QUERYACTIVECOUNT message

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nCount = (nuint16)wParam;
pnArray = (pnint)lParam;
```

Parameters

nCount

Specifies the number of items contained in the array specified by the *pnArray* parameter.

pnArray

Specifies an array of integers to be filled.

Remarks

nCount corresponds to the value the MDI child window returned in the NWA_MSG_STATUSBAR_QUERYACTIVECOUNT message.

The MDI child window should fill each element of the array in the NWA_MSG_STATUSBAR_NEWACTIVEITEMS message with a unique identifier for each status bar item added upon receiving the NWA_MSG_STATUSBAR_ADDPREFITEM message. The elements in the array should be filled in the order they are found on the status bar.

See Also

NWA_MSG_STATUSBAR_NEWACTIVEITEMS, NWA_MSG_STATUSBAR_QUERYACTIVECOUNT

NWA_MSG_STATUSBAR_QUERYITEMCOUNT

Is sent to an MDI child window when the user opens the status bar preferences dialog box
Platform: Windows 3.1
Service: Snap-in

Syntax

```
#include <nwsnapin.h>
```

Return Values

The view should return the count of status bar items.

Remarks

The MDI child window responds to the NWA_MSG_STATUSBAR_QUERYITEMCOUNT message by returning the number of items to be added to the status bar preferences dialog box. If the MDI child window returns a value greater than zero, the NetWare Administrator main window will respond by sending the indicated number of NWA_MSG_STATUSBAR_ADDPREFITEM messages back to the MDI child window.

The dialog box allows the user to determine which items will actually be visible in the status bar.

See Also

NWA_MSG_STATUSBAR_ADDPREFITEM

NWA_MSG_STATUSBAR_SETITEMTEXT

Replaces the text in a previously defined status bar text field with the contents of the *pszText* parameter

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = (WPARAM)itemId;
lParam = (LPARAM)pszText;
```

Parameters

wParam

Specifies the unique identifier of the text field to be replaced.

lParam

Points to the NULL-terminated string to replace the contents of the text field in the status bar.

Remarks

The *itemId* parameter corresponds to the *itemId* parameter of the NWAStatusBarItemStruct structure passed in the *lParam* parameter of the NWA_MSG_STATUSBAR_ADDITEM message.

NWA_MSG_TOOLBAR_ADDITEM

Is sent to the top MDI child window after it sends a NWA_MSG_TOOLBAR_POPULATE message to the NetWare Administrator main window

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = (WPARAM)hwndView;
lParam = (LPARAM)pToolBarItem;
```

Parameters

hwndView

Specifies the window handle of the view.

pToolBarItem

Points to the NWAToolBarItemStruct structure containing the data required to add a button or spacer to the toolbar.

Return Values

NWA_RET_SUCCESS
NWA_ERR_ERROR

Remarks

To insert a separator to the toolbar, *pToolBarItem* must be set to TDIB_SEPARATOR rather than the address of the NWAToolBarItemStruct structure.

If the item was successfully added to the toolbar, NWA_RET_SUCCESS will be returned by the NWA_MSG_TOOLBAR_ADDITEM message; otherwise, NWA_ERR_ERROR will be returned.

The items will appear on the toolbar in the order they are added. The items should be added in the order indicated by the values returned by the NWA_MSG_TOOLBAR_NEWACTIVEITEMS message.

If an item is added whose *menuId* matches a previous items *menuId*, the previous item will be removed from the status bar. In general, the last item to be added to the status bar with a duplicate *menuId* will be added to the status bar while the previous item with a matching *menuId* will be deleted from the status bar.

See Also

NWA_MSG_TOOLBAR_NEWACTIVEITEMS

NWA_MSG_TOOLBAR_ADDPREFITEM

Is sent by the NetWare Administrator main window to the view window according to the number returned by the NWA_MSG_TOOLBAR_QUERYITEMCOUNT message

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nPrefCnt = (nuint16)wParam;
ptbPrefItem = (NWAToolBarPrefItemStruct*)lParam;
```

Parameters

nPrefCnt

Specifies the sequence number of the preference items.

ptbPrefItem

Points to the NWAToolBarPrefItemStruct structure containing the data necessary to display an item in the toolbar preferences dialog box.

Remarks

The view window will fill in the NWAToolBarPrefItemStruct structure.

The results will be displayed in the toolbar preferences dialog box which allows you to select which toolbar items will be visible in the toolbar.

The items will appear in the toolbar preference dialog box in the order they are added.

NWA_MSG_TOOLBAR_GETBUTTONSTATE

Is sent from an MDI child window to the NetWare Administrator main window to determine the state of the button corresponding to the button ID passed in *menuId*

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = (WPARAM)menuId;
lParam = 0;
```

Parameters

menuId

Specifies the unique identifier of the toolbar button (corresponds to the *menuId* structure field of the NWAToolBarItemStruct structure.

Return Values

NWA_MSG_TOOLBAR_GETBUTTONSTATE returns the state of the button (TBUp, TBDown, or TBIndeterminate).

NWA_MSG_TOOLBAR_GETBUTTONTYPE

Is sent by a view to the NetWare Administrator main window to determine what type of button corresponds to the button ID passed in *menuId*

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = (WPARAM)menuId;
lParam = 0;
```

Parameters

menuId

Specifies the unique identifier of the toolbar button (corresponds to the *menuId* structure field of the NWAToolBarItemStruct structure.

lParam

Is not currently being used; must be zero.

Return Values

NWA_MSG_TOOLBAR_GETBUTTONTYPE returns the button type (TBCommand, TBToggle, or TBUnknown).

NWA_MSG_TOOLBAR_NEWACTIVEITEMS

Is sent from the NetWare Administrator main window to a view when the user has pressed OK in the toolbar preferences dialog box

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nCount = (nuint16)wParam;
pnArray = (pnint)lParam;
```

Parameters

nCount

Specifies the number of active items selected by the user.

pnArray

Points to an array of *nCount* integers. If the value of is zero, the *pnArray* parameter will be NULL.

Remarks

The NWA_MSG_TOOLBAR_NEWACTIVEITEMS message indicates the user's toolbar preferences have changed.

Each integer value in the array pointed to by the *pnArray* parameter is the *menuId* of an item that should be active in the toolbar. If the value is TBID_SEPARATOR, a separator should be added to the toolbar.

The order of the items in the array pointed to by the *pnArray* parameter indicates the preferred order of items in the views portion of the toolbar.

Before the NetWare Administrator main window sends a NWA_MSG_TOOLBAR_NEWACTIVEITEMS message to the view, it will depopulate the toolbar. After NetWare Administrator handles the NWA_MSG_TOOLBAR_NEWACTIVEITEMS message, the view will receive a NWA_MSG_TOOLBAR_POPULATE message.

The view is responsible for saving and restoring its own preferences.

See Also

NWA_MSG_TOOLBAR_QUERYACTIVECOUNT,
NWA_MSG_TOOLBAR_QUERYACTIVEITEMS

NWA_MSG_TOOLBAR_POPULATE

Is sent to the top MDI child window each time the toolbar is created

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

hwndMain = (HWND) wParam;
```

Parameters

hwndMain

Specifies the NetWare Administrator main window sending the NWA_MSG_TOOLBAR_POPULATE message (value of the *wParam* parameter.

Remarks

When a window receives the NWA_MSG_TOOLBAR_POPULATE message, it can respond by sending a NWA_MSG_TOOLBAR_ADDITEM message to *hwndMain*.

NWA_MSG_TOOLBAR_QUERYACTIVECOUNT

Is sent to an MDI child window when the user opens the toolbar preferences dialog box

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>
```

Remarks

The MDI child window responds to the NWA_MSG_TOOLBAR_QUERYACTIVECOUNT message by returning the number of items that will be in the active items list in the toolbar pereferences dialog box.

If the MDI child window returns a value greater than zero, the NetWare Administrator main window will respond by sending a NWA_MSG_TOOLBAR_QUERYACTIVEITEMS message. The NWA_MSG_TOOLBAR_QUERYACTIVEITEMS message will provide the address of any array of integers whose size corresponds to the value returned by the NWA_MSG_TOOLBAR_QUERYACTIVECOUNT message.

The dialog box allows the user to determine which items will be visible in the toolbar.

See Also

NWA_MSG_TOOLBAR_NEWACTIVEITEMS, NWA_MSG_TOOLBAR_QUERYACTIVEITEMS

NWA_MSG_TOOLBAR_QUERYACTIVEITEMS

Is sent to a view when the user opens the toolbar preferences dialog box and after the view responds to a NWA_MSG_TOOLBAR_QUERYACTIVECOUNT message

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nCount = (nuint16)wParam;
pnArray = (pnint)lParam;
```

Parameters

nCount

Specifies the number of active items to be displayed in the toolbar preferences dialog box.

pnArray

Points to an array of *nCount* integers (which correspond to the menu ID of a button).

Remarks

The number specified by the *nCount* parameter corresponds to the value the view returned in the NWA_MSG_TOOLBAR_QUERYACTIVECOUNT message.

The view should fill each element of the array pointed to by the *pnArray* parameter with the number specified by the *menuId* parameter of the active toolbar item. The elements should be filled in the order they are found on the toolbar.

TDIB_SEPARATOR can be used to indicate the presence of a separator on the toolbar.

See Also

NWA_MSG_TOOLBAR_NEWACTIVEITEMS,
NWA_MSG_TOOLBAR_QUERYACTIVECOUNT

NWA_MSG_TOOLBAR_QUERYENABLEBUTTON

Is sent to an MDI child window when the NetWare Administrator main window wants to know if a toolbar item should be enabled or disabled (grayed out)

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

nMenuId = (nint)wParam;
```

Parameters

nMenuId

Specifies the *menuId* corresponding to the toolbar button.

Return Values

N_TRUE
N_FALSE

Remarks

The NWA_MSG_TOOLBAR_QUERYENABLEBUTTON message will return N_TRUE if the button should be enabled; otherwise it will return N_FALSE.

NWA_MSG_TOOLBAR_QUERYITEMCOUNT

Is sent to an MDI child window when the user opens the toolbar preferences dialog box

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>
```

Return Values

The MDI child window should return the number of items to be added to the toolbar preferences dialog box.

Remarks

The MDI child window responds to the NWA_MSG_TOOLBAR_QUERYITEMCOUNT message by returning the number of items it will add to the toolbar preferences dialog box. If the MDI child window returns a value greater than zero, the NetWare Administrator main window will respond by sending the indicated number of NWA_MSG_TOOLBAR_ADDPREFITEM messages back to the MDI child window.

The dialog box allows the user to determine which items will be visible in the toolbar.

See Also

NWA_MSG_TOOLBAR_ADDPREFITEM

NWA_MSG_TOOLBAR_SETBUTTONSTATE

Is sent by an MDI child window to the NetWare Administrator main window to set the state of the button corresponding to the button ID passed in the *wParam* parameter

Platform: Windows 3.1

Service: Snap-in

Syntax

```
#include <nwsnapin.h>

wParam = (WPARAM)menuId;
lParam = (LPARAM)buttonState;
```

Parameters

menuId

Specifies the unique identifier of the toolbar button.

buttonState

Specifies the desired state of the button (TBUp, TBDown, or TBIndeterminate).

Remarks

The *menuId* parameter corresponds to the *menuId* structure field of the NWAToolBarItemStruct structure.

See Also

NWA_MSG_TOOLBAR_GETBUTTONSTATE

NWA_WM_CANCLOSE

Receives this message when OK is selected and is the dialog procedure for a page dialog in a MPEW

Syntax

```
#include <nwsnapin.h>

DialogProc(HWND hdlg, UINT msg, WPARAM wParam, LPARAM lParam);
```

Parameters

hdlg

Specifies the handle of the dialog box.

msg

Specifies NWA_WM_CANCLOSE.

p1 and *p2*

Are not currently being used.

Return Values

TRUE
FALSE

Remarks

If TRUE is returned, the MPEW window will close. If FALSE is returned, the MPEW window will not close.

NWA_WM_F1HELP

Receives this message when F1 (Help) is selected and is the dialog procedure for a page dialog in MPEW

Syntax

```
#include <nwsnapin.h>

DialogProc(HWND hdlg, UINT msg, WPARAM wParam, LPARAM lParam);
```

Parameters

hdlg

Specifies the handle of the dialog box.

msg

Specifies NWA_WM_F1HELP.

p1 and *p2*

Are not currently being used.

NWA_WM_SETPAGEMODIFY

Should be set to the page dialog window to indicate the page dialog information has changed

Syntax

```
#include <nwsnapin.h>

DialogProc(HWND hdlg, UINT msg, WPARAM wParam, LPARAM lParam);
```

Parameters

hdlg

Specifies the handle of the dialog box.

msg

Specifies NWA_WM_SETPAGEMODIFY.

p1 and *p2*

Are not currently being used.

Remarks

If TRUE is sent in the low high-order word of the *1Param* parameter of the **SendMessage** function, the page has been modified.

If FALSE is sent, it has not been modified.

Time/Date Manipulation

Time/Date Manipulation: Functions

asctime, asctime_r

Converts the time information into a string
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

char *asctime (
   const struct tm  *timeptr);

#include <time.h>

char *asctime_r (
  const struct tm  *timeptr,
  char             *string);
```

Parameters

timeptr

(IN) Specifies the structure containing the time information to convert.

string

(OUT) The converted string.

Return Values

Return a pointer to the character string result.

Remarks

asctime and **asctime_r** convert the time information in the structure pointed to by *timeptr* into a string containing exactly 26 characters.

This string has the form shown in the following example:

```
Wed Mar 21 15:58:27 1990\n\0
```

All fields have a constant width. The newline character \n and the NULL character (\0) occupy the last two positions of the string. The area containing the returned string is reused each time **asctime** is called.

acetime has the same function as **acetime_r**, except that **acetime_r**

requires the caller to pass storage for the results rather than relying on a global variable to store the results. **asctime_r** is supported only in CLIB V 4.11 or above.

See Also

clock, **ctime**, **ctime_r**, **difftime**, **gmtime**, **gmtime_r**, **localtime**, **localtime_r** , **mktime**, **strftime**, **time**

Example

asctime

```
#include <stdio.h>
#include <time.h>

main ()
{
   struct tm   *time_of_day;
   time_t      ltime;
   time (&ltime);
   time_of_day = localtime (&ltime);
   printf ("Date and time is: %s.Get to work.", asctime (time_of day));
}
```

produces the following:

```
Date and time is: Wed Mar 21 15:58:27 1990
.Get to work.
```

clock

Returns the number of hundredths of seconds since the NLM™ application began executing

Local Servers: nonblocking

Remote Servers: N/A

Classification: ANSI

SMP Aware: Yes

Service: Time/Date Manipulation

Syntax

```
#include <time.h>

clock_t clock (void);
```

Return Values

The number of hundredths of seconds is returned encoded into type clock_t.

Remarks

It is often important to run benchmarks or measure the time it takes for a process to execute. The following example shows how to call **clock** to time a process down to 1/100 of a second. In the example, assume that CLK_TCK is a constant equal to 100.

asctime_r requires the user to pass storage for the function result, rather than relying on a global variable for the result as in **asctime**.

See Also

time

_ConvertDOSTimeToCalendar

Converts DOS-style date and time to calendar time
Local Servers: nonblocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: Yes
Service: Time/Date Manipulation

Syntax

```
#include <nwtime.h>

time_t  _ConvertDOSTimeToCalendar (
   LONG   dateTime);
```

Parameters

dateTime
(IN) Specifies the DOS-style date and time to be converted.

Return Values

Returns the calendar time.

Remarks

_ConvertDOSTimeToCalendar converts DOS-style date and time used in directory entries (used in the DIR structure) to calendar time.

The standard DOS date and time format is as follows:

Calendar time represents the time in seconds since January 1, 1970 (UTC).

See Calendar program: Example.

See Also

_ConvertTimeToDOS

_ConvertTimeToDOS

Converts calendar time to DOS-style date and time
Local Servers: nonblocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: Yes
Service: Time/Date Manipulation

Syntax

```
#include <nwdos.h>

void _ConvertTimeToDOS (
   time_t            calendarTime,
   struct _DOSDate  *fileDate,
   struct _DOSTime  *fileTime);
```

Parameters

calendarTime

(IN) Specifies the calendar time to be converted to DOS-style date and time.

fileDate

(OUT) Specifies the DOS-style date.

fileTime

(OUT) Specifies the DOS-style time.

Return Values

None

Remarks

This function converts calendar time to DOS-style date and time used in directory entries (used in the DIR structure).

The standard DOS date and time format is as follows:

Calendar time represents the time in seconds since January 1, 1970 (UTC).

See Also

_ConvertDOSTimeToCalendar

ctime, ctime_r

Converts the calendar time to local time in the form of a string
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

char *ctime (
   const time_t  *timer);

#include <time.h>

char *ctime_r (
   const time_t  *timer,
   char          *string);
```

Parameters

timer

(IN) Specifies the calendar time to convert to local time.

string

(OUT) The converted string.

Return Values

Returns the pointer to the string containing the local time.

Remarks

ctime, ctime_r converts the calendar time information in the*timer* structure into a local-time string containing 26 characters.

The string has the form shown in the following example:

```
Wed Mar 21 15:58:27 1990\n\0
```

The string is equivalent to:

```
asctime (localtime (timer) )
```

All fields have a constant width. The newline character (\n) and the

NULL character (\0) occupy the last two positions of the string. The area containing the returned string is reused each time **ctime, ctime_r** is called.

ctime_r requires the caller to pass storage for the results rather than relying on a global variable to store the results. **ctime_r** is supported only in CLIB V 4.11 or above.

See Also

asctime, asctime_r, clock, difftime, gmtime, gmtime_r, localtime, localtime_r, mktime, strftime, time

Example

ctime

```
#include <stdio.h>
#include <time.h>

void print_time ()
{
   time_t time_of_day;
   time_of_day = time (NULL);
   printf ("It is now: %s.Get to work.\n", ctime (&time_of_day) );
}
```

produces the following:

```
It is now: Tue Dec 25 15:58:42 1990
.Get to work.
```

GetClockStatus

Returns the time-of-day clock and status register
Local Servers: blocking
Remote Servers: N/A
Classification: 4.x
SMP Aware: Yes
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

void GetClockStatus (
   clockAndStatus   *dataPtr);
```

Parameters

dataPtr
(OUT) Receives the clock registers and status.

Return Values

None

Remarks

GetClockStatus succeeds whether or not time synchronization services are active. You must examine the status register to determine whether the time synchronization services are active and whether the clock is actually synchronized. There are three masks which can be used to determine the status of the clock.

CLOCK_SYNCHRONIZATION_IS_ACTIVE implies that the time synchronization NLM is loaded and active. If clock synchronization is not active, then the default state is to set CLOCK_IS_SYNCHRONIZED and clear CLOCK_IS_NETWORK_SYNCHRONIZED. If clock synchronization is not active, then the other status bits have no meaning. However, accepting the default state of CLOCK_IS_SYNCHRONIZED and taking precautions to prevent the exchange of time dependent information with other servers on the network might be acceptable to many applications.

CLOCK_IS_NETWORK_SYNCHRONIZED implies that the clock can be used for network timestamps when CLOCK_IS_SYNCHRONIZED is also true. If CLOCK_IS_NETWORK_SYNCHRONIZED is not set, then time synchronization is inactive or the server has become isolated from other servers on the network. In this case, any timestamp derived from the current clock should not be used to coordinate events with other servers.

CLOCK_IS_SYNCHRONIZED implies that the time can safely be used for timestamps for operations only on the local server. When CLOCK_IS_NETWORK_SYNCHRONIZED is also set, then timestamps can be used to coordinate events with other servers that are synchronized to network time.

The parameter type clockAndStatus is defined as

```
typedef LONG clockAndStatus[3];
```

fields of the data are:

dataPtr [0]	whole seconds
dataPtr [1]	fractional seconds
dataPtr [2]	status bits

Together the whole and fractional seconds fields can be interpreted as a 64-bit binary number with a binary point separating the two fields. During each clock interrupt the fractional portion is incremented by a value determined from the basic clock interrupt frequency of the host machine and by a value supplied by time synchronization services to account for drift between different clocks on the network. Attempts to use the fractional seconds as a high precision timer fail because the implied precision far exceeds the actual precision of any known hardware clock and the increment values are not guaranteed to be uniform.

The whole seconds field represents the time (in seconds) since January 1, 1970, and is reported as Universal Coordinated Time (previously known as GMT or Greenwich Mean Time) rather than local time. This format is compatible with that returned by the ANSI **time** function.

GetCurrentTicks

Returns the current server up-time
Local Servers: nonblocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <nwstring.h>

LONG GetCurrentTicks (void);
```

Return Values

Returns the current server up-time in ticks (approximately eighteenths of a second).

See Also

clock, **time**

Example

GetCurrentTicks

```
#Include <nwstring.h>

LONG   serverUpTime;
serverUpTime = GetCurrentTicks ();
```

gmtime, gmtime_r

Converts calendar time into Coordinated Universal Time (UTC)
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

struct tm *gmtime (
   const time_t  *timer);

#include <time.h>

struct tm *gmtime_r (
  const time_t   *timer,
  struct tm      *timeP);
```

Parameters

timer

(IN) Specifies the calendar time to convert to UTC.

timeP

(OUT) Pointer to the converted string.

Return Values

Returns a pointer to a structure containing the broken-down time.

Remarks

gmtime, gmtime_r converts the calendar time pointed to by the *timer* parameter into a broken-down time, expressed as UTC.

Use the NetWare® console command SET TIME to set the date and time kept by the server. Refer to the *NetWare 386 System Administration* manual for more information on this command.

The structure returned is reused every time that **gmtime, gmtime_r** is called.

gmtime_r requires the caller to pass storage for the results rather than relying on a global variable to store the results. **gmtime_r** is supported only in CLIB V 4.11 or above.

See Also

asctime, asctime_r, **clock**, **ctime, ctime_r**, **difftime**, **localtime, localtime_r**, **mktime**, **strftime**, **time**

localtime, localtime_r

Converts calendar time to local time
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

struct tm *localtime (
   const time_t  *timer);

#include <time.h>

struct tm *localtime_r (
   const time_t  *timer,
   struct tm     *timeP);
```

Parameters

timer

(IN) Specifies the calendar time to convert into the tm structure.

timeP

(OUT) Pointer to the converted string.

Return Values

Returns a pointer to the tm structure containing the time information.

Remarks

localtime, localtime_r converts the calendar time pointed to by the *timer* parameter into a structure of time information expressed as local time.

The calendar time (Coordinated Universal Time) is usually obtained by calling the **time** function.

Use the NetWare console command **SET TIME** to set the date and time kept by the server. Refer to the *NetWare 386 System Administration* manual for more information on this command.

The structure returned is reused every time that **localtime** is called.

localtime_r requires the caller to pass storage for the results rather than relying on a global variable to store the results.**localtime_r** is supported only in CLIB V 4.11 or above.

See Also

asctime, asctime_r, **clock**, **ctime, ctime_r**, **difftime**, **gmtime, gmtime_r**, **mktime**, **strftime**, **time**

Example

localtime

```
#include <stdio.h>
#include <time.h>

void print_time ()
{
   time_t time_of_day;
   time_of_day = time (NULL);
   printf ("It is now: %s.Get to work.",asctime (localtime (&time_of_da
}
```

produces the following:

```
It is now: Wed Mar 21 15:58:27 1990
.Get to work.
```

mktime

Converts the time information in a structure into calendar time
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

time_t mktime (
   struct tm  *timeptr);
```

Parameters

timeptr

(IN) Specifies the structure containing the time information to convert.

Return Values

Returns the converted calendar time. On error, -1 will be returned cast as type (time_t). To test for an error condition, compare the return value to (time_t) -1.

Remarks

mktime converts the time information in the structure pointed to by the *timeptr* parameter into a calendar time with the same encoding used by the **time** function.

The original values of the *tm_sec*, *tm_min*, *tm_hour*, *tm_mday*, and *tm_mon* fields are not restricted to ranges described for the tm structure. If these fields are not in their proper ranges, they are adjusted so that they are in the proper ranges. Values for the *tm_wday* and *tm_yday* fields are computed after all the other fields have been adjusted.

See Also

asctime, asctime_r, **clock**, **ctime, ctime_r**, **difftime**, **gmtime, gmtime_r**, **localtime, localtime_r**, **strftime**, **time**

Example

mktime

```
#include <stdio.h>
#include <time.h>

static const char *week_day[ ] =
{
   "Sunday", "Monday", "Tuesday", "Wednesday",
   "Thursday", "Friday", "Saturday"
};

main ()
{
   struct tm new_year;
   new_year.tm_year = 2001 - 1900;
   new_year.tm_mon  = 0;
   new_year.tm_mday = 1;
   new_year.tm_hour = 0;
   new_year.tm_min  = 0;
   new_year.tm_sec  = 0;
   new_year.tm_isdst = 0;
   mktime (&new_year);
   printf ("The next century begins on a %s\n",
      week_day [new_year.tm_wday] );
}
```

produces the following:

```
The next century begins on a Monday
```

NWPackDateTime

Packs a date and time into an nuint32
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

nuint32 N_API NWPackDateTime (
   NW_DATE N_FAR  *sDate,
   NW_TIME N_FAR  *sTime);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWPackDateTime
  (Var sDate : NW_DATE;
   Var sTime : NW_TIME
) : nuint32;
```

Parameters

sDate
(IN) Points to the NW_DATE structure (optional).

sTime
(IN) Points to the NW_TIME structure (optional).

Return Values

Returns the packed date and time upon successful completion.

Remarks

NWPackDateTime returns the packed date and time. If a parameter is NULL, the associated bits will be set to zero.

Many functions return dates in a packed format identical to that defined by DOS. Time occupies the low order word and date occupies the high order word. The bits are defined as follows:

```
0-4     seconds divided by two
5-10    minutes
11-15   hours (0-23)
16-20   day
21-24   month
25-31   year minus 1980
```

NWPackDateTime does no validity checking on the passed information. The programmer should ensure dates and/or times in the associated structures are valid before passing them to **NWPackDateTime**.

NCP Calls

None

See Also

NWUnpackDateTime

NWUnpackDateTime

Unpacks a packed date and time into the passed structures
NetWare Server: 2.2, 3.11, 3.12, 4.x
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Time/Date Manipulation

Syntax

```
#include <nwmisc.h>
or
#include <nwcalls.h>

void N_API NWUnpackDateTime (
   nuint32          dateTime,
   NW_DATE N_FAR  *sDate,
   NW_TIME N_FAR  *sTime);
```

Pascal Syntax

```
#include <nwmisc.inc>

Procedure NWUnpackDateTime
  (dateTime : nuint32;
   Var sDate : NW_DATE;
   Var sTime : NW_TIME
);
```

Parameters

dateTime
(IN) Specifies the date and time in packed format.

sDate
(OUT) Points to the NW_DATE structure (optional).

sTime
(OUT) Points to the NW_TIME structure (optional).

Return Values

None

Remarks

Many functions return dates in a packed format identical to that defined by DOS. The time occupies the low order word and the date occupies the high order word. The bits are defined as follows:

```
0-4     Seconds divided by two
5-10    Minutes
11-15   Hours (0-23)
16-20   Day
21-24   Month
25-31   Year minus 1980
```

NCP Calls

None

See Also

NWPackDateTime

SecondsToTicks

Converts seconds to clock ticks
Local Servers: nonblocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: Yes
Service: Time/Date Manipulation

Syntax

```
#include <nwstring.h>

void SecondsToTicks (
   LONG   seconds,
   LONG   tenthsOfSeconds,
   LONG  *ticks);
```

Parameters

seconds

(IN) Specifies the number of seconds to convert.

tenthsOfSeconds

(IN) Specifies the tenths of seconds to convert.

ticks

(OUT) Receives the equivalent number of clock ticks.

Return Values

None.

Remarks

To convert clock ticks back to seconds, call the **TicksToSeconds** function.

One IBM* PC clock tick is approximately 1/18 second. (Eighteen [18.21] clock ticks equal approximately 1 second.)

See Also

TicksToSeconds

strftime

Formats the time into an array under format control
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

size_t strftime (
   char              *s,
   size_t             maxsize,
   const char        *format,
   const struct tm   *timeptr);
```

Parameters

s

(IN) Specifies a character array.

maxsize

(IN) Specifies the maximum number of characters that can be placed in the array.

format

(IN) Specifies the format control string.

timeptr

(IN) Specifies the time argument.

Return Values

If the number of characters to be placed into the array is less than the value specified by the *maxsize* parameter, the number of characters placed into the array pointed to by the *s* parameter (not including the terminating NULL character) will be returned. Otherwise, a value of 0 is returned. If an error occurs, *errno* is set.

Remarks

strftime formats the time in the *timeptr* parameter into the array pointed to by the *s* parameter according to the *format* parameter.

The *format* parameter string consists of zero or more directives and ordinary characters. A directive consists of a character followed by a character that determines the substitution that is to take place. All ordinary characters are copied unchanged into the array. No more than the number of characters specified by the *maxsize* parameter are placed in the array.

%a	Locale's abbreviated weekday name
%A	Locale's full weekday name
%b	Locale's abbreviated month name
%B	Locale's full month name
%c	Locale's appropriate date and time representation
%d	Day of the month as a decimal number (01-31)
%D	Date in the format mm/dd/yy (POSIX)
%h	Locale's abbreviated month name (POSIX)
%H	Hour (24-hour clock) as a decimal number (00-23)
%I	Hour (12-hour clock) as a decimal number (01-12)
%j	Day of the year as a decimal number (001-366)
%m	Month as a decimal number (01-12)
%M	Minute as a decimal number (00-59)
%n	Newline character (POSIX)
%p	Locale's equivalent of either AM or PM
%r	12-hour clock time (01-12) using the AM/PM notation in the format hh:mm:ss (POSIX)
%S	Second as a decimal number (00-59)
%t	Tab character (POSIX)
%T	24-hour clock time in the format hh:mm:ss (POSIX)
%U	Week number of the year as a decimal number (00-52) where Sunday is the first day of the week
%w	Weekday as a decimal number (0-6) where 0 is Sunday
%W	Week number of the year as a decimal number (00-52) where Monday is the first day of the week
%x	Locale's appropriate date representation
%X	Locale's appropriate time representation
%y	Year without century as a decimal number (00-99)
%Y	Year with century as a decimal number
%Z	Timezone name, or by no characters if no timezone exists
%%	Character %

See Also

asctime, asctime_r, clock, ctime, ctime_r, difftime, gmtime, gmtime_r, localtime, localtime_r, mktime, setlocale, time

Example

strftime

```
#include <stdio.h>
#include <time.h>

main()
{
   time_t time_of_day;
   char buffer[80];
   time_of_day = time (NULL);
   strftime (buffer, 80, "Today is %A %B %d, %Y",
      localtime (&time_of_day) );
   printf ("%s\n", buffer);
}
```

produces the following:

```
Today is Friday December 25, 1990
```

TicksToSeconds

Converts clock ticks to seconds
Local Servers: nonblocking
Remote Servers: N/A
Classification: 2.x, 3.x, 4.x
SMP Aware: Yes
Service: Time/Date Manipulation

Syntax

```
#include <nwstring.h>

void TicksToSeconds (
   LONG   Ticks,
   LONG  *seconds,
   LONG  *tenthsOfSeconds);
```

Parameters

ticks
(IN) Specifies the number of ticks to convert to seconds.

seconds
(OUT) Specifies the number of seconds.

tenthsOfSeconds
(OUT) Specifies the tenths of seconds.

Return Values

None

Remarks

TicksToSeconds converts clock ticks to seconds and tenths of seconds. To convert seconds back to clock ticks, call the**SecondsToTicks** function.

One IBM PC clock tick is approximately 1/18 second. (Eighteen [18.21] clock ticks equal approximately 1 second.)

See Also

SecondsToTicks

time

Returns the current calendar time
Local Servers: nonblocking
Remote Servers: N/A
Classification: ANSI
SMP Aware: Yes
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

time_t time (
   time_t  *timer);
```

Parameters

timer
(IN) Specifies the object containing the encoded calendar time.

Return Values

Returns the current calendar time encoded into type time_t. On error, returns -1 cast as type (time_t). To test for an error condition, compare the return value to (time_t) -1.

Remarks

The time represents the time in seconds since January 1, 1970 (Universal Coordinated Time). If the *timer* parameter is not NULL, the current calendar time is also stored in the object pointed to by the *timer* parameter.

See Also

clock

tzset

Sets the *tzname*, *timezone*, and *daylight* global variable parameters
Local Servers: nonblocking
Remote Servers: N/A
Classification: Other
SMP Aware: No
Service: Time/Date Manipulation

Syntax

```
#include <time.h>

void tzset (void);
```

Return Values

None

Remarks

Before accessing the *tzname*, *timezone*, and *daylight* global variable parameters, you must first call **tzset** to initialize or update these variables to the correct values.

CLIB.NLM uses the time zone information that is set by issuing the **SET TIMEZONE** (also **SET TIME ZONE**) console command.

For NetWare 3.x, **tzset** always uses the time zone information that is active when CLIB.NLM loads. If the **SET TIMEZONE** command is issued after CLIB.NLM loads, **tzset** does not use the updated information. Therefore, for NetWare 3.x, the **SET TIMEZONE** command must be issued in the following prescribed order:

At the server console, issue the **SET TIMEZONE** command with the time zone name, hours, and daylight savings code parameters.

Load CLIB.NLM.

Load your NLM application.

Refer to the *Supervising the Network* manual for more information on the **SET TIMEZONE** command.

In NetWare 4.x, **tzset** uses the current time zone setting of the server to initialize or update the values of the *tzname*, *timezone*, and *daylight* global variable parameters. However, remember that these variables will not be automatically updated when the **SET TIMEZONE** command is issued. They are updated only when **tzset** is called. For this reason, you should call **tzset** before calling any of the time functions. Doing so ensures that the *tzname*, *timezone*, and *daylight* global variable parameters contain the proper values.

See Also

asctime, asctime_r, **gmtime, gmtime_r**, **localtime, localtime_r**, **time**

Example

tzset

```
#include <time.h>
#include <stdio.h>

void print_zone()
{
   tzset();

   printf( "time zone names: %s %s\ n", tzname[0], tzname[1]);
   printf( "timezone: %ld \ n", timezone ) ;
   printf( "daylight: %d\n", daylight ) ;
}
```

produces the following:

```
time zone names: PST PDT
timezone: 28800
daylight: 1
```

Time/Date Manipulation: Structures

NW_DATE

Contains the date in packed format

Service: Time/Date Manipulation

Defined In: nwmisc.h and nwmisc.inc

Structure

```
typedef struct {
   nuint8    day;
   nuint8    month;
   nuint16   year;
} NW_DATE;
```

Pascal Structure

```
NW_DATE = Record
    day : nuint8;
    month : nuint8;
    year : nuint16
  End;
```

Fields

day

Specifies the day.

month

Specifies the month.

year

Specifies the year.

NW_TIME

Contains the time in packed format

Service: Time/Date Manipulation

Defined In: nwmisc.h and nwmisc.inc

Structure

```
typedef struct {
   nuint8    seconds;
   nuint8    minutes;
   nuint16   hours;
} NW_TIME;
```

Pascal Structure

```
NW_TIME = Record
    seconds : nuint8;
    minutes : nuint8;
    hours : nuint16
  End;
```

Fields

seconds

Specifies the seconds.

minutes

Specifies the minutes.

hours

Specifies the hours.

Remarks

Because the *hours* field is defined as a nuint16, NW_TIME takes up a nuint32 of space.

tm

Contains time information

Service: Time/Date Manipulation

Defined In: time.h

Structure

```
struct tm {
   int   tm_sec;
   int   tm_min;
   int   tm_hour;
   int   tm_mday;
   int   tm_mon;
   int   tm_year;
   int   tm_wday;
   int   tm_yday;
   int   tm_isdst;
};
```

Fields

tm_sec

Contains the number of seconds after the minute. This number is in the range [0,61].

tm_min

Contains the number of minutes after the hour. This number is in the range [0,59].

tm_hour

Contains the number of hours after midnight. This number is in the range [0,23].

tm_mday

Contains the day of the month. This number is in the range [1,31].

tm_mon

Contains the months since January. This number is in the range [0,11].

tm_year

Contains the number of years since 1900.

tm_wday

Contains the number of days since Sunday. This number is in the range [0,6].

tm_yday

Contains the number of days since January 1. This number is in the range [0,365].

tm_isdst

Contains a Daylight Savings Time flag, defined as follows:

0	Daylight savings time is not in effect.
>0	Daylight savings time is in effect.
<0	Daylight savings time information is not available.

Remarks

Some locales such Korean, Chinese, and Italian use the *tm_wday* structure field for the standard date format. If the *tm_wday* structure field is not set, an incorrect day will be displayed. An application which sets only the year, month, and day structure fields can compute the weekday by calling the C library function:

```
mktime(timePtr)
```

TTS

TTS: Guides

TTS: General Guide

Tasks

Enabling TTS

Concepts

TTS Introduction

Implicit Transaction Tracking

Explicit Transaction Tracking

Transaction Tracking Process

Implicit Tracking Threshold

TTS Transaction Functions

TTS Status and File Control Functions

TTS Threshold Functions

TTS: Concepts

TTS: Functions

TTS: Structures

Parent Topic:

Management Overview

TTS: Tasks

Enabling TTS

When TTS is disabled, NDS operations which require modifying the database on that server are also disabled.

1. **At the console prompt of the file server, type ENABLE TTS.**

   NOTE: If TTS was disabled because volume SYS: was full, log onto the server and delete unnecessary files from volume SYS:, then type ENABLE TTS at the console.

Parent Topic:

TTS: General Guide

TTS: Concepts

Explicit Transaction Tracking

Explicit transaction tracking uses TTS. To signal an explicit transaction, bracket file update sequences with a pair of TTS functions.

NWTTSBeginTransaction signals the beginning of a transaction.

NWTTSEndTransaction signals the end of a transaction.

TTS tracks all transactional files accessed between these two functions and automatically places a physical lock on transactional files that you write to during this time. As with implicit tracking, if your connection is disrupted, TTS backs out of any modified files. You can force TTS to back out of the transaction by calling **NWTTSAbortTransaction**.

TTS affects only transactional files. To make a file transactional, set the file's transaction bit. This bit is a file attribute and can be modified using the File System service. (See File System Directory Entry Attributes.)

Parent Topic:

TTS: General Guide

Related Topics:

Implicit Transaction Tracking

Implicit Tracking Threshold

TTS lets you read and modify the implicit tracking threshold. This is the number of logical and physical locks your application can set before implicit tracking takes effect. The default threshold is 0, meaning that whenever you place a logical or physical lock on a transactional file, the file will be tracked. A threshold of 0xFF means that implicit transactions for the associated lock type have been completed.

There are two reasons to modify the threshold value:

If your application is performing explicit transactions but also locking records you don't want to track, you can turn off implicit transactions.

If your application always keeps one or more records locked, raising the threshold prevents these records from being tracked.

Parent Topic:

TTS: General Guide

Implicit Transaction Tracking

Implicit transaction tracking requires no coding on your part. If TTS is installed and enabled on a NetWare® server, transaction tracking applies to any logical or physical record lock you place on a transactional file. If the workstation's connection is disrupted before the records are unlocked, TTS backs out of the transaction and restores the records to their original state.

Parent Topic:

TTS: General Guide

Related Topics:

Explicit Transaction Tracking

Transaction Tracking Process

A typical transaction scenario is a banking database application that writes a debit to one account, a credit to another account, and a note to a log. The dependencies among the three operations make it extremely important that they are performed as a single transaction.

If any one of the operations fails, the integrity of the other operations is undermined. TTS ensures data integrity under circumstances such as these. Below is a description of how TTS responds to a request to write to a transactional file.

1. TTS stores the new data in cache memory. The file itself remains unchanged.

2. TTS scans the target file on the server hard disk, finds the data to be changed (old data), and copies the old data to cache memory. TTS also records the name and directory path of the target file and the location and length of the old data (record) within the file. The target file on the server hard disk is still unchanged.

3. TTS writes the old data in cache memory to a transaction work file on the server hard disk. The transaction work file resides at the root level of a volume on the server. The file is flagged system and hidden. The target file on the server hard disk is still unchanged.

4. TTS finally writes the new data in cache memory to the target file on the server hard disk.

TTS repeats these steps for each write within a transaction. The transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, TTS writes the contents of the transaction work file to the target file, thereby restoring the file to its pre-transaction state.

Parent Topic:

TTS: General Guide

TTS Introduction

The Transaction Tracking System™ (TTS™) software allows NetWare® servers to track transactions and ensure file integrity by backing out of or erasing interrupted or partially completed transactions. For example, the server can back out of a transaction if your application terminates unexpectedly while a transaction is in progress.

A transaction can include any series of requests that affect transactional files. TTS can monitor from 1 to 200 transactions at a time. The maximum number (50 to 200) is configurable.

TTS can track only one transaction at a time for each session. If a session sends several transactions to a server rapidly, TTS queues the transactions and services them one at a time. TTS lets you begin and end transactions, monitor TTS status, and access TTS information.

For a description of structures and other data definitions relating to this chapter, see TTS: Structures.

Parent Topic:

TTS: General Guide

TTS Status and File Control Functions

These functions check and modify the status of TTS on a NetWare® server and access the TTS transaction bit flags associated with transactional files.

Function	Header	Comment
NWDisableTTS	nwtts.h	Disables transaction tracking on a server.
NWEnableTTS	nwtts.h	Enables transaction tracking on a server.
NWGetTTSStats	nwtts.h	Returns TTS statistics.
NWTTSGetControlFlags	nwtts.h	Returns the transaction bits for files flagged as transactional.
NWTTSIsAvailable	nwtts.h	Verifies that the server supports transaction tracking.

NWTTSSetControlFlags	nwtts.h	Enables or disables automatic record locking when writing to transactional files.
NWTTSTransactionStatus	nwtts.h	Verifies whether a transaction has been written to disk.

Parent Topic:

TTS: General Guide

TTS Threshold Functions

These functions read and modify the connection and process thresholds affecting implicit transaction tracking.

Function	Header	Comment
NWTTSGetConnectionThresholds	nwtts.h	Returns the number of logical and physical record locks allowed before implicit transactions begin.
NWTTSGetProcessThresholds	nwtts.h	Returns the number of explicit physical and logical record locks allowed before implicit locking begins.
NWTTSSetConnectionThresholds	nwtts.h	Informs a server of how many explicit physical and logical record locks to permit before invoking implicit transactions.
NWTTSSetProcessThresholds	nwtts.h	Sets the number of logical and physical locks to perform before implicit locking begins.

Parent Topic:

TTS: General Guide

TTS Transaction Functions

These functions perform transaction tracking.

Function	Header	Comment
NWTTSAbortTransaction	nwtts.h	Aborts all transactions, explicit or implicit. When this function returns successfully, all transactions have been successfully backed out of.
NWTTSBeginTransaction	nwtts.h	Begins an explicit transaction.
NWTTSEndTransaction	nwtts.h	Ends an explicit transaction and returns a transaction reference number.

Parent Topic:

TTS: General Guide

TTS: Functions

NWDisableTTS

Disables transaction tracking on a NetWare® server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12
Platform: DOS, NLM, OS/2, Windows* 3.1, Windows NT*, Windows*95
Service: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWDisableTTS (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWDisableTTS
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

Transaction Tracking is always enabled on 4.x servers due to Directory

Services requirements; therefore, enabling or disabling transaction tracking is only supported on 2.x and 3.x servers.

NCP Calls

0x2222 23 207	Disable Transaction Tracking

NWEnableTTS

Enables transaction tracking on a NetWare server
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2, 3.11, 3.12
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWEnableTTS (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWEnableTTS
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

Transaction Tracking is always enabled on 4.x servers due to Directory Services requirements; therefore, enabling or disabling transaction tracking is only supported on 2.x and 3.x servers.

NCP Calls

0x2222 23 208   Enable Transaction Tracking

NWGetTTSStats

Returns Transaction Tracking System™ (TTS™) statistics
Local Servers: blocking
Remote Servers: blocking
NetWare Server: 2.2
Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95
Service: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWGetTTSStats (
   NWCONN_HANDLE       conn,
   TTS_STATS NWPTR   ttsStats);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWGetTTSStats
  (conn : NWCONN_HANDLE;
   Var ttsStats : TTS_STATS
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

ttsStats

(OUT) Points to TTS_STATS where the TTS statistics will be returned.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89C6	NO_CONSOLE_PRIVILEGES

Remarks

You must have console rights to call**NWGetTTSStats**.

NCP Calls

0x2222 23 213   Get Transaction Tracking Statistics

NWTTSAbortTransaction

Aborts all transactions, explicit and implicit
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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSAbortTransaction (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSAbortTransaction
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values for more information.

0x00	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89FE	DIRECTORY_LOCKED
0x89FE	Transaction Restart
0x89FF	LOCK_ERROR

Remarks

When **NWTTSAbortTransaction** is complete, all transactions will have been successfully backed out.

If a transaction is aborted, all Writes made since the beginning of a transaction are cancelled, and all files are returned to the state they were in before the transaction began.

NWTTSAbortTransaction releases the following record locks:

Physical record locks generated by the NetWare server when an application tried to write an unlocked record.

Physical or logical locks not released because of a file Write.

0x89FE indicates more than the threshold number of logical or physical records are still locked by the application. However, the transaction is finished and any locks being held are released. When this happens, the NetWare server automatically starts a new implicit transaction.

NCP Calls

0x2222 34 3   TTS Abort Transaction

See Also

NWTTSBeginTransaction, **NWTTSEndTransaction**

NWTTSBeginTransaction

Begins an explicit transaction
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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSBeginTransaction (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSBeginTransaction
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn
(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values for more information.

0x0000	Successful
0x8801	INVALID_CONNECTION

Remarks

NWTTSBeginTransaction tracks all transactional files currently open, and those opened during the transaction.

When data is written to a transaction file during a transaction, the NetWare server automatically generates a physical record lock for the region being written. If a lock already exists, no additional lock is generated. This automatic locking can be disabled by calling **NWTTSSetControlFlags**.

Any closing and unlocking of transaction files is delayed until either **NWTTSEndTransaction** or **NWTTSAbortTransaction** is executed. Logical and physical records are not unlocked until the end of the transaction if file writes are performed while the lock is in force.

NCP Calls

0x2222 34 1   TTS Begin Transaction

See Also

NWTTSAbortTransaction, **NWTTSEndTransaction**, **NWTTSSetControlFlags**

NWTTSEndTransaction

Ends an explicit transaction and returns a transaction reference number
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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSEndTransaction (
   NWCONN_HANDLE   conn,
   pnuint32        transactionNum);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSEndTransaction
   (conn : NWCONN_HANDLE;
    transactionNum : pnuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

transactionNum

(OUT) Points to the transaction reference number for the transaction being ended (optional).

Return Values

These are common return values; see Return Values for more information.

0x0000	Successful
0x8801	INVALID_CONNECTION
0x89FE	Transaction Restart

0x89FF	LOCK_ERROR

Remarks

The transaction number is used to verify a successful transaction completion to disk.

The transaction is not necessarily written to disk when the reference number is returned. A client must call **NWTTSTransactionStatus** to verify a transaction has been written to disk. If the NetWare server fails before all updates contained within the transaction have been written to disk, the transaction is backed out when the NetWare server is rebooted.

If transaction tracking is disabled, *transactionNum* can still determine when the transaction has been completely written to disk. Since *transactionNum* is optional, substitute NULL if no return values are desired.

NWTTSEndTransaction releases all physical record locks generated by the NetWare server when a Write is made to an unlocked record. In addition, physical or logical locks that were not released due to a file Write are unlocked at this time.

0x89FE indicates more than the threshold number of logical or physical records are still locked by the application. However, the transaction is finished and any locks being held are released. In this case, the NetWare server automatically starts a new implicit transaction.

NCP Calls

0x2222 34 2   TTS End Transaction

See Also

NWTTSAbortTransaction, **NWTTSBeginTransaction**, **NWTTSTransactionStatus**

NWTTSGetConnectionThresholds

Returns the number of logical and physical record locks allowed before implicit transactions begin

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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSGetConnectionThresholds (
   NWCONN_HANDLE   conn,
   pnuint8         logicalLockLevel,
   pnuint8         physicalLockLevel);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSGetConnectionThresholds
   (conn : NWCONN_HANDLE;
    logicalLockLevel : pnuint8;
    physicalLockLevel : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

logicalLockLevel

(OUT) Points to the number of logical record locks allowed before implicit transactions begin (0 to 255, optional).

physicalLockLevel

(OUT) Points to the number of physical record locks allowed before implicit transactions begin (0 to 255, optional).

Return Values

These are common return values; see Return Values for more

information.

0x0000	Successful
0x8801	INVALID_CONNECTION

Remarks

Both **NWTTSSetConnectionThresholds** and **NWGetConnectionThreshold**s are useful for applications changing the implicit application thresholds that later want to restore them. For example, **NWTTSGetConnectionThresholds** can get the number of logical and physical locks, and **NWTTSSetConnectionThresholds** can perform one of the following:

Turn off implicit transactions. (Applications using only explicit transactions, but sometimes generating unnecessary implicit transactions, need to turn off all implicit transactions.)

Set implicit thresholds for applications always keeping one or more records locked.

The default threshold for logical and physical locks is 0. 0xFF means implicit transactions for the lock type have been completed.

Both *physicalLockLevel* and *logicalLockLevel* are optional parameters. Substitute NULL if these parameters are not to be returned. However, all parameter positions must be filled.

NCP Calls

0x2222 34 7   TTS Get Workstation Threshold

See Also

NWTTSSetConnectionThresholds

NWTTSGetControlFlags

Returns the transaction bits for files flagged as transactional

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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSGetControlFlags (
   NWCONN_HANDLE   conn,
   pnuint8         controlFlags);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSGetControlFlags
  (conn : NWCONN_HANDLE;
   controlFlags : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

controlFlags

(OUT) Points to Transaction Tracking Control flags.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

Transaction tracking control flags are only valid for files flagged as TTS. Bits 1 to 7 in *controlFlags* are reserved; bit 0 is defined below:

```
0x00   Automatic record locking is disabled
0x01   Automatic record locking is enabled
```

NCP Calls

0x2222 34 9   TTS Get Transaction Bits

See Also

NWTTSSetControlFlags

NWTTSGetProcessThresholds

Returns the number of explicit physical and logical record locks allowed before implicit locking begins

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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSGetProcessThresholds (
   NWCONN_HANDLE   conn,
   pnuint8         logicalLockLevel,
   pnuint8         physicalLockLevel);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSGetProcessThresholds
   (conn : NWCONN_HANDLE;
    logicalLockLevel : pnuint8;
    physicalLockLevel : pnuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

logicalLockLevel

(OUT) Points to the number of explicit logical record locks allowed before implicit transactions begin (0 to 255, optional).

physicalLockLevel

(OUT) Points to the number of explicit physical record locks allowed before implicit transactions begin (0 to 255, optional).

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NWTTSGetProcessThresholds and **NWTTSSetProcessThresholds** are useful for applications changing the implicit process thresholds that later want to restore them. For example, **NWTTSGetProcessThresholds** can query an application for the number of logical and physical record locks allowed before an implicit transaction begins, and **NWTTSSetProcessThresholds** can perform one of the following:

Turn off implicit transactions. (Applications intending to use only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.)

Set implicit thresholds for applications always keeping one or more records locked.

The default threshold for logical and physical locks is 0. 0xFF means no implicit transactions are allowed for the lock type.

Thresholds returned by **NWTTSGetProcessThresholds** are valid for the requesting application only. When the application terminates, the connection thresholds are restored.

Both *physicalLockLevel* and *logicalLockLevel* are optional parameters. Substitute NULL if these parameters are not to be returned. However, all parameter positions must be filled.

NCP Calls

0x2222 34 5 TTS Get Application Threshold

See Also

NWTTSSetProcessThresholds

NWTTSIsAvailable

Verifies the NetWare server supports transaction tracking

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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSIsAvailable (
   NWCONN_HANDLE   conn);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSIsAvailable
  (conn : NWCONN_HANDLE
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

Return Values

These are common return values; see Return Values for more information.

0x0000	Transaction Tracking is Unavailable
0x8801	INVALID_CONNECTION
0x89FD	Transaction Tracking is Disabled
0x89FF	Transaction Tracking is Available

Remarks

0x0000 does not indicate successful completion of **NWTTSIsAvailable**. Instead, 0x0000 indicates TTS is unavailable. The successful completion code is 0x89FF, TTS is available.

NCP Calls

0x2222 34 0   TTS Is Available

NWTTSSetConnectionThresholds

Sets the number of explicit physical and logical record locks to permit before invoking implicit transactions

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSSetConnectionThresholds (
   NWCONN_HANDLE   conn,
   nuint8          logicalLockLevel,
   nuint8          physicalLockLevel);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSSetConnectionThresholds
   (conn : NWCONN_HANDLE;
    logicalLockLevel : nuint8;
    physicalLockLevel : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

logicalLockLevel

(IN) Specifies the number of logical record locks to allow before implicit transactions begin (0 to 255).

physicalLockLevel

(IN) Specifies the number of physical record locks to allow before implicit transactions begin (0 to 255).

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

The return values are in effect for all applications, not just ones calling **NWTTSSetConnectionThresholds**.

The default threshold for logical and physical locks is 0. 0xFF means no implicit transactions for the lock type can be performed.

NWTTSSetConnectionThresholds and **NWTTSGetConnectionThresholds** are useful for applications changing the implicit application thresholds that later want to restore them.

For example, **NWTTSGetConnectionThresholds** can obtain the current number of logical and physical locks, and **NWTTSSetConnectionThresholds** can perform one of the following:

Turn off implicit transactions. (Applications using only explicit transactions, but sometimes generate unnecessary implicit transactions, need to turn off all implicit transactions.)

Set implicit thresholds for applications always keeping one or more records locked.

NCP Calls

0x2222 34 8	TTS Set Workstation Thresholds

See Also

NWTTSGetConnectionThresholds

NWTTSSetControlFlags

Enables or disables automatic record locking on Writes to transactional files

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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSSetControlFlags (
   NWCONN_HANDLE   conn,
   nuint8          controlFlags);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSSetControlFlags
  (conn : NWCONN_HANDLE;
   controlFlags : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

controlFlags

(IN) Specifies the Transaction Tracking control flags.

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION

Remarks

NWTTSSetControlFlags allows a client to set the transaction bits in *controlFlag*.

Transaction tracking control flags are only valid for files flagged as transactional. Only bit 0 is used currently. Flag definitions follow:

```
0x00   Automatic record locking is disabled
0x01   Automatic record locking is enabled
```

NCP Calls

0x2222 34 10   TTS Set Transaction Bits

NWTTSSetProcessThresholds

Sets the number of logical and physical locks to perform before implicit locking begins

Local Servers: blocking

Remote Servers: blocking

NetWare Server: 2.2, 3.11, 3.12, 4.x

Platform: DOS, NLM, OS/2, Windows 3.1, Windows NT, Windows95

Service: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSSetProcessThresholds (
   NWCONN_HANDLE   conn,
   nuint8          logicalLockLevel,
   nuint8          physicalLockLevel);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSSetProcessThresholds
   (conn : NWCONN_HANDLE;
    logicalLockLevel : nuint8;
    physicalLockLevel : nuint8
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

logicalLockLevel

(IN) Specifies the number of logical record locks to allow before implicit transactions begin (0-255).

physicalLockLevel

(IN) Specifies the number of physical record locks to allow before implicit transactions begin (0-255).

Return Values

These are common return values; see Return Values for more

information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x8996	SERVER_OUT_OF_MEMORY

Remarks

The thresholds set by **NWTTSSetProcessThresholds** are valid for the requesting application only. When the application terminates, the default workstation thresholds are restored.

NWTTSSetProcessThresholds either turns off implicit transactions or allows applications always keeping one or more records locked to work. Applications intending to use only explicit transactions, but sometimes generating unnecessary implicit transactions, can call **NWTTSSetProcessThresholds** to turn off all implicit transactions.

The default threshold for logical and physical locks is 0 unless the number has been changed by calling **NWTTSSetConnectionThresholds**. 0xFF means no implicit transactions for the lock type are performed.

NCP Calls

0x2222 34 6   TTS Set Application Thresholds

See Also

NWTTSGetProcessThresholds

NWTTSTransactionStatus

Verifies whether a transaction has been written to disk

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: Transaction Tracking System (TTS)

Syntax

```
#include <nwtts.h>
or
#include <nwcalls.h>

NWCCODE N_API NWTTSTransactionStatus (
   NWCONN_HANDLE   conn,
   nuint32         transactionNum);
```

Pascal Syntax

```
#include <nwtts.inc>

Function NWTTSTransactionStatus
  (conn : NWCONN_HANDLE;
   transactionNum : nuint32
) : NWCCODE;
```

Parameters

conn

(IN) Specifies the NetWare server connection handle.

transactionNum

(IN) Specifies the transaction reference number (obtained from **NWTTSEndTransaction**).

Return Values

These are common return values; see Return Values for more information.

0x0000	SUCCESSFUL
0x8801	INVALID_CONNECTION
0x89FF	Transaction not written to disk

Remarks

NWTTSTransactionStatus can be called even if **NWTTSEndTransaction** returns TTS_DISABLED.

Applications should not wait for transactions to be written to disk unless it is absolutely necessary. Because the NetWare server caches algorithms, it may be up to five seconds before they are actually written. Transactions are written to disk in the order in which they terminate.

NCP Calls

0x2222 34 4	TTS Transaction Status

See Also

NWTTSEndTransaction

TTS: Structures

TTS_STATS

Returns TTS statistics

Service: TTS

Defined In: nwtts.h

Structure

```
typedef struct
{
   nuint32   systemElapsedTime;
   nuint8    TTS_Supported;
   nuint8    TTS_Enabled;
   nuint16   TTS_VolumeNumber;
   nuint16   TTS_MaxOpenTransactions;
   nuint16   TTS_MaxTransactionsOpened;
   nuint16   TTS_CurrTransactionsOpen;
   nuint32   TTS_TotalTransactions;
   nuint32   TTS_TotalWrites;
   nuint32   TTS_TotalBackouts;
   nuint16   TTS_UnfilledBackouts;
   nuint16   TTS_DiskBlocksInUse;
   nuint32   TTS_FATAllocations;
   nuint32   TTS_FileSizeChanges;
   nuint32   TTS_FilesTruncated;
   nuint8    numberOfTransactions;
   struct
   {
      nuint8   connNumber;
      nuint8   taskNumber;
   } connTask[235];
} TTS_STATS;
```

Pascal Structure

```
Defined in nwtts.inc

 TTS_STATS = Record
    systemElapsedTime : nuint32;
    TTS_Supported : nuint8;
    TTS_Enabled : nuint8;
    TTS_VolumeNumber : nuint16;
    TTS_MaxOpenTransactions : nuint16;
    TTS_MaxTransactionsOpened : nuint16;
    TTS_CurrTransactionsOpen : nuint16;
    TTS_TotalTransactions : nuint32;
    TTS_TotalWrites : nuint32;
    TTS_TotalBackouts : nuint32;
    TTS_UnfilledBackouts : nuint16;
```

```
        TTS_DiskBlocksInUse : nuint16;
        TTS_FATAllocations : nuint32;
        TTS_FileSizeChanges : nuint32;
        TTS_FilesTruncated : nuint32;
        numberOfTransactions : nuint8;
        connTask : Array[0..234] Of CONN_TASK;
      End;

    CONN_TASK = Record
        connNumber : nuint8;
        taskNumber : nuint8;
      End;
```

Fields

systemElapsedTime

TTS_Supported

TTS_Enabled

TTS_VolumeNumber

TTS_MaxOpenTransactions

TTS_MaxTransactionsOpened

TTS_CurrTransactionsOpen

TTS_TotalTransactions

TTS_TotalWrites

TTS_TotalBackouts

TTS_UnfilledBackouts

TTS_DiskBlocksInUse

TTS_FATAllocations

TTS_FileSizeChanges

TTS_FilesTruncated

numberOfTransactions