

# **NNTP-32 Library**

**Network News Transfer Protocol  
Dynamic Link Library  
for Microsoft® Windows™**

**Version 5.2**

**Copyright © 1994 - 2003 by Distinct Corporation  
All rights reserved**



## **Table of Contents**

Table of Contents.....	iii
<b>1</b> Overview .....	5
<b>1.1</b> Introduction .....	5
<b>1.2</b> Registry Entries .....	5
<b>1.3</b> Function Summary .....	6
<b>2</b> Reference .....	7
<b>2.1</b> nntp_article () .....	7
<b>2.2</b> nntp_date () .....	9
<b>2.3</b> nntp_direct () .....	10
<b>2.4</b> nntp_forward () .....	11
<b>2.5</b> nntp_fw_open () .....	12
<b>2.6</b> nntp_group () .....	15
<b>2.7</b> nntp_help () .....	16
<b>2.8</b> nntp_list () .....	17
<b>2.9</b> nntp_new () .....	18
<b>2.10</b> nntp_open () .....	20
<b>2.11</b> nntp_option () .....	22
<b>2.12</b> nntp_position () .....	24
<b>2.13</b> nntp_post_buf () .....	25
<b>2.14</b> nntp_post () .....	26
<b>2.15</b> nntp_quit () .....	27
<b>2.16</b> nntp_reply () .....	28
<b>2.17</b> nntp_slave () .....	29
<b>2.18</b> nntp_xopen () .....	30
<b>2.19</b> nntp_xquit () .....	32
<b>2.20</b> nntp_yield () .....	33



## 1 Overview

### 1.1 Introduction

The Distinct NNTP-32 (Network News Transfer Protocol) library provides developers with an API to navigate through all available network news groups on a server and to retrieve or post news articles.

Once a connection has been established with a server using the `nntp_(x)open` call, a current news group must be selected with the `nntp_group` command. A complete list of all news groups supported by the server can be obtained with the `nntp_list` function. Within the current news group, articles or portions of articles can be retrieved with the `nntp_article` call. An application can post its own news articles with calls to `nntp_post`.

When a connection is no longer needed, it must be closed with a call to `nntp_quit`. Connections should not be kept open when not in use since they consume valuable system resources. An application must close all open connections before it terminates.

Most functions in the NNTP-32 library wait for a reply from the NNTP server before returning to the caller. Because communication links or servers sometimes go down, these functions use a default timeout of 20 seconds before returning the error value `NNTP_TIMED_OUT` if no response was received. This timeout value can be changed by editing or creating a registry entry as described in the section '1.2 - Registry Entries'.

While waiting for a reply to a command or while waiting for a data transfer to complete, most functions in the NNTP-32 library yield control to Windows. This allows applications to process messages while an NNTP operation is in progress. Without yielding, the system would appear to be frozen for the entire duration of each NNTP operation. By default, yielding is therefore enabled but can be disabled if necessary with the `nntp_option` call. Yielding control to Windows is done by the NNTP-32 library by frequently calling the `nntp_yield` function during lengthy operations. This function is exported in case the calling application also needs to yield, for example while processing data in a callback function. The calling application can also define its own yielding function to be called instead of the default yielding function (`nntp_yield` may then be called from within the applications yielding function).

### 1.2 Registry Entries

Various parameters used to control the behavior of the Distinct NNTP-32 library are stored under the following registry path.

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Distinct\DLLS\NNTP32**

The following entries specify various Distinct NNTP-32 library settings.

**Timeout**                      **REG\_DWORD**                      *1-999*

Specifies the default timeout in seconds used while connecting to the remote NNTP server and for other NNTP operations. The default value is 20 seconds.

**DebugLevel**                      **REG\_DWORD**                      *0-2*

Specifies what debug information should be logged into the D32-NNTP.DBG file in the directory from which D32-NNTP.DLL was loaded. The following values are supported.

Value	Meaning
0	Disable logging.
1	Log all errors.
2	Log all function calls and return values.

A higher debug level automatically includes all lower debug levels. The default value is 0.

### 1.3 Function Summary

**nntp\_article**

Retrieve all or a portion of an article or get article statistics

**nntp\_date**

Get current time and date formatted as specified in RFC 850

**nntp\_direct**

Send a command over NNTP connection

**nntp\_forward**

Forward news article to next server

**nntp\_fw\_open**

Open connection with NNTP server via firewall

**nntp\_group**

Select current news group

**nntp\_help**

Retrieve remote help instructions

**nntp\_list**

Retrieve list of available news groups

**nntp\_new**

Retrieve list of new groups or new articles

**nntp\_open**

Open connection with NNTP server

**nntp\_option**

Set connection option

**nntp\_position**

Set current article

**nntp\_post\_buf**

Post news article

**nntp\_post**

Post news article

**nntp\_quit**

Disconnect from NNTP server

**nntp\_reply**

Get last reply of NNTP server

**nntp\_slave**

Inform server of slave status

**nntp\_xopen**

Asynchronous version of nntp\_open

**nntp\_xquit**

Cancel asynchronous connect or operation in progress

**nntp\_yield**

Yield control to Windows

## 2 Reference

### 2.1 nntp\_article ()

**Description** Retrieve all or a portion of an article or get article statistics.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_article (hConn, id, num, flags, proc)
```

**GLOBALHANDLE** *hConn*; Handle of NNTP structure from **nntp\_(x)open**

**LPSTR** *id*; Name of article

**unsigned long** *num*; Number of article

**int** *flags*; Type of access

**FARPROC** *proc*; Application callback function

**Remarks** The **nntp\_article** function retrieves all of or a portion of the news article or gets statistics on the news article with the name pointed to by *id* or with the number specified by *num*. If *num* is set to zero and *id* points to an empty string then the current article will be accessed. The type of access is specified with the *flags* parameter which must be set to one of the following.

ARTICLE_ALL	Entire article
ARTICLE_BODY	Message portion of article
ARTICLE_HEAD	Article header
ARTICLE_STAT	Article statistics

The **nntp\_article** function downloads a message from the server by calling the application supplied callback function *proc* with message data. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
```

```
char far *buf;
```

```
int len;
```

```
DWORD lParam;
```

An application can abort the transfer of a message by returning -1 in the callback function (all other return values are currently ignored) or by calling the **nntp\_xquit** function. Note that aborting a message transfer will also abort the underlying connection by resetting it. If the transfer was aborted successfully, then the **nntp\_article** call will return NNTP\_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **nntp\_quit**) can be made until a new connection has been established with **nntp\_(x)open**.

Once the message has been transferred, one final call to *proc* is made with the *len* parameter set to zero to indicate the end of the message.

If the *flags* parameter was set to ARTICLE\_STAT, then the callback function *proc* is not called. Instead, the application must call **nntp\_reply** to retrieve the statistics about the article.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_article** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.2 nntp\_date ()

**Description** Get current time and date formatted as specified in RFC 850.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_date (date)
```

```
LPSTR date;           Buffer for formatted date string
```

**Remarks** The `nntp_date` function builds a string specifying the current time and date in RFC 850 format. This string can then be used to build the required date field in an NNTP message header.

**Return Value** The `nntp_date` function returns the length of the string placed into the buffer pointed to by *date*.

## 2.3 nntp\_direct ()

**Description** Send a command over NNTP connection.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_direct (hConn, msg, rep_buf, len)
```

<b>GLOBALHANDLE</b> <i>hConn</i> ;	Handle of NNTP structure from <b>nntp_(x)open</b>
<b>LPSTR</b> <i>msg</i> ;	Command to send
<b>char</b> * <i>rep_buf</i> ;	Buffer for command response
<b>int</b> * <i>len</i> ;	Length of response buffer

**Remarks** This function sends the command specified by *msg* to the NNTP server. The response returned by the server is placed into the buffer pointed to by *rep\_buf*. The size of the response buffer is specified by *len*. The return buffer is NULL terminated. If *rep\_buf* is not large enough for the entire response then the message from the NNTP server will be truncated and the *len* parameter will contain the required length of the buffer.

Note: This function is supplied to allow the user to send any commands that are not supported by the NNTP Library. This method is not a substitute for the existing functions and should not be used as such. In particular this function should not be used to disconnect as the NNTP Library will not be aware of the action. This function cannot be used until a successful connection has been made with **nntp\_open** or **nntp\_fw\_open**.

**Return Value** The **nntp\_direct** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

NNTP\_INSUFFICIENT\_LENGTH

The reply buffer is not large enough to hold the entire response, the *len* parameter will contain the required length.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.4 nntp\_forward ()

**Description** Forward news article to next server.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_forward (hConn, id, proc)
```

<b>GLOBALHANDLE</b> <i>hConn</i> ;	Handle of NNTP structure from <b>nntp_(x)open</b>
<b>LPSTR</b> <i>id</i> ;	Name of article
<b>FARPROC</b> <i>proc</i> ;	Application callback function

**Remarks** The **nntp\_forward** function forwards the news article identified by *id* to the current server. The **nntp\_forward** function uploads an article to the server by calling the application supplied callback function *proc* requesting message data. The *proc* callback is called until it returns zero to indicate the end of the message. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

An application can abort forwarding of an article by returning -1 in the callback function or by calling the **nntp\_xquit** function. Note that aborting forwarding will also abort the underlying connection by resetting it. If the forwarding was aborted successfully, then the **nntp\_forward** call will return NNTP\_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **nntp\_quit**) can be made until a new connection has been established with **nntp\_(x)open**.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

The **nntp\_forward** function is normally only used by NNTP servers.

**Return Value** The **nntp\_forward** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.5 nntp\_fw\_open ()

**Description** Open connection with NNTP server via the firewall.

```
#include <windows.h>
#include <d32-nntp.h>
#include <d32-fw.h>
```

```
int FAR PASCAL nntp_fw_open (conn, proc, lParam, fw)
```

<b>GLOBALHANDLE</b> far *conn;	Buffer for handle of NNTP structure
<b>FARPROC</b> proc;	Application callback function
<b>DWORD</b> lParam;	Application supplied parameter
<b>fw_param</b> far *fw	A pointer to the fw_param firewall structure

**Remarks** The **nntp\_fw\_open** function establishes a TCP/IP connection with the specified host via the firewall.

After a successful connection, the handle of the connection control structure is placed into the buffer pointed to by *conn*. This handle must be used in all subsequent calls to the other NNTP functions for this connection. The Distinct NNTP Library supports multiple concurrent NNTP connections limited only by system resources.

During development it may be necessary to observe exactly what commands are sent to the NNTP server and what replies are generated. For this purpose, the application can specify the address of a debug callback function with *proc*. Note that *proc* must be obtained by calling **MakeProcInstance** specifying an exported function. This function will receive a copy of all network traffic and must be defined as follows.

```
int FAR PASCAL (*proc) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

The *proc* parameter must be set to NULL if the application does not want to receive debug messages.

The *lParam* parameter can be set to any value that has meaning to the application such as an index into an array or a pointer to a structure. This parameter is passed back to the application as an argument in each callback function and allows the application to pass connection specific data to the callback function.

The *fw* parameter is a pointer to the **fw\_param** firewall structure which must contain all the connection specific fields required to establish connection to the NNTP server via the firewall. The **fw\_param** structure has the following fields.

```
typedef struct fw_parameters
{
    char far * methods;                different types of methods.
    unsigned char n_methods;          the number of methods
    unsigned char add_type;           address type of the destination address
    char * domain_name;              the domain name
    char far * fw_host;               IP address of the firewall server
    char far * dest_host;             IP address of the SMTP server
    unsigned short fw_port;           firewall port
    unsigned short dest_port;         destination port
    char far * username;              valid user id
    char far * passwd;                user password
    int socks_ver;                    the socks version;
```

```

    char reserved [256 - 6 * sizeof(char far *) - 2 * sizeof(unsigned char) - 2 * sizeof(unsigned
    short) - sizeof(int)];
} fw_param;

```

Distinct firewall library supports both SOCKS version 4 and version 5 firewall server. The *socks\_ver* field of the **fw\_param** structure should specify the SOCKS version. The *socks\_ver* field should be any or a combination of the following values:

FW_VERSION5	The Socks version is 5
FW_VERSION4	The Socks version is 4

If the *socks\_ver* field contains both FW\_VERSION5 and FW\_VERSION4 then the Distinct firewall library will try to automatically determine the firewall server version and send appropriate information to the firewall server to establish a connection.

The *methods* field can point to string "0", "1", or "2" or any combination of "0", "1", "2", for example "01" or "12". Method 0 means no authentication is required, 1 means GSSAPI and 2 means a *username* and *passwd* is required. *n\_methods* is the number of methods that appear in the *method* field. Note that only methods 0 and 2 are supported currently and if method 2 is specified then the application must specify a *username* and *passwd* for authentication. Note that the *methods*, *n\_methods*, and *passwd* fields are required only if the application is trying to connect to a version 5 firewall server.

The field *add\_type* specifies the type of address of the remote host. The *add\_type* can be any one of the following types.

Type	Meaning
FW_ADDR_IP4	address is a version 4 IP address
FW_ADDR_DNS	address is a DNS style domain name
FW_ADDR_IP6	address is a version 6 IP address

If the *addr\_type* field is of type FW\_ADDR\_DNS then the *domain\_name* field must point to a DNS-style domain name. If the value of the *addr\_type* field is FW\_ADDR\_IP4 or FW\_ADDR\_IP6 then the *dest\_host* field must contain the IP address of the destination host. Note that if the SOCKS version is 4 then the *add\_type* field should always be FW\_ADDR\_IP4 or FW\_ADDR\_DNS.

The *fw\_host* field should contain the address of the firewall server. The port of the firewall server must be specified in the *fw\_port* field, if this field is 0 then the firewall library will use the default port 1080. The field *dest\_port* must contain port number of the destination host. Both *fw\_port* and the *dest\_port* must be in host byte order.

The *username* should contain a valid user id if the application is trying to connect to a SOCKS version 4 firewall server or if the application has specified method 2 for authentication in case of SOCKS version 5.

Note that any string pointer of the **fw\_param** structure that is not used should be set to NULL.

**Return Value** The **nntp\_fw\_open** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_FAILURE

Remote error.

NNTP\_CANNOT\_INITIALIZE

Support libraries or network drivers could not be initialized.

NNTP\_CANNOT\_INIT\_WINSOCK

Windows Sockets transport library could not be initialized.

NNTP\_OUT\_OF\_MEMORY

Not enough memory to allocate connection control structure.

NNTP\_UNKNOWN\_HOST

Unable to resolve specified host name.

NNTP\_CANNOT\_ALLOC\_SOCKET

Windows Sockets library could not allocate a socket.

NNTP\_CANNOT\_BIND\_SOCKET

Windows Sockets library was not able to bind socket to local port.

NNTP\_HOST\_NOT\_RESPONDING

Unable to connect to NNTP port on specified host.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

If this function returns `NNTP_FAILURE`, then the `nntp_reply` function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.6 nntp\_group ()

**Description** Select current news group.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_group (hConn, group)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
```

```
LPSTR group;              Name of news group
```

**Remarks** The **nntp\_group** function sets the current news group.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_group** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.7 nntp\_help ()

**Description** Retrieve remote help instructions.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_help (hConn, proc)
```

```
GLOBALHANDLE hConn;    Handle of NNTP structure from nntp_(x)open
FARPROC proc;          Application callback function
```

**Remarks** The **nntp\_help** function retrieves a list of help instructions from the server. These messages are delivered to the application by calling the application supplied callback function *proc* for each line of help information. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, line, len, lParam)

GLOBALHANDLE hConn;
char far *line;
int len;
DWORD lParam;
```

An application can abort the transfer of help instructions by returning -1 in the callback function (all other return values are currently ignored) or by calling the **nntp\_xquit** function. Note that aborting help instructions will also abort the underlying connection by resetting it. If the help instructions were aborted successfully, then the **nntp\_help** call will return NNTP\_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **nntp\_quit**) can be made until a new connection has been established with **nntp\_(x)open**.

Once the messages have been transferred, one final call to *proc* is made with the *len* parameter set to zero to indicate the end of the messages.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_help** function either returns NNTP\_SUCCESS or one of the following errors.

```
NNTP_INVALID_HANDLE
    Invalid connection handle was specified.

>NNTP_FAILURE
    Host was not able to execute command.

>NNTP_CANNOT_SEND_COMMAND
    Windows Sockets library was not able to accept outgoing data.

>NNTP_TIMED_OUT
    Time out occurred while waiting for a response from host.

>NNTP_IN_PROGRESS
    Another operation is already in progress.

>NNTP_ABORTED
    Operation was aborted and connection has been closed.
```

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.8 nntp\_list ()

**Description** Retrieve list of available news groups.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_list (hConn, proc)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
FARPROC proc;           Application callback function
```

**Remarks** The **nntp\_list** function retrieves a list of available news groups from the server. The news groups are delivered to the application by calling the application supplied callback function *proc* for each news group. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, line, len, lParam)

GLOBALHANDLE hConn;
char far *line;
int len;
DWORD lParam;
```

An application can abort the listing by returning -1 in the callback function (all other return values are currently ignored) or by calling the **nntp\_xquit** function. Note that aborting a listing will also abort the underlying connection by resetting it. If the listing was aborted successfully, then the **nntp\_list** call will return NNTP\_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **nntp\_quit**) can be made until a new connection has been established with **nntp\_(x)open**.

Once the news groups have been transferred, one final call to *proc* is made with the *len* parameter set to zero to indicate the end of the callbacks.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_list** function either returns NNTP\_SUCCESS or one of the following errors.

```
NNTP_INVALID_HANDLE
    Invalid connection handle was specified.

>NNTP_FAILURE
    Host was not able to execute command.

>NNTP_CANNOT_SEND_COMMAND
    Windows Sockets library was not able to accept outgoing data.

>NNTP_TIMED_OUT
    Time out occurred while waiting for a response from host.

>NNTP_IN_PROGRESS
    Another operation is already in progress.

>NNTP_ABORTED
    Operation was aborted and connection has been closed.
```

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.9 nntp\_new ()

**Description** Retrieve list of new groups or new articles.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_new (hConn, flags, cmdline, group, proc)
```

<b>GLOBALHANDLE</b> <i>hConn</i> ;	Handle of NNTP structure from <b>nntp_(x)open</b>
<b>int</b> <i>flags</i> ;	Type of access
<b>LPSTR</b> <i>cmdline</i> ;	Command line specifying date and time
<b>LPSTR</b> <i>group</i> ;	Group name for NEW_NEWS
<b>FARPROC</b> <i>proc</i> ;	Application callback function

**Remarks**

The **nntp\_new** function retrieves a list of new news groups or new news articles from the server. The *flags* parameter must be set to one of the following two values to select the type of information to be retrieved.

NEW_GROUPS	Get new news groups.
NEW_NEWS	Get new news articles.

The *cmdline* parameter must contain the required date and time arguments and may also contain optional arguments to select GMT and to specify distributions. The *cmdline* syntax is

**date time [GMT] [distributions]**

The **date** argument is required and must be given as 6 digits in the format YYMMDD, where YY specifies the last two digits of the year, MM the two digits of the month (with leading zero if less than 10) and DD the day of the month (with leading zero if less than 10). The **time** argument is also required and must be given as 6 digits in the format HHMMSS, where HH specifies the hours 00 through 23, MM the minutes 00 through 59 and SS the seconds 00 through 59. The time is assumed to be in the servers time zone unless the optional **GMT** argument is specified. For example, setting the *cmdline* argument to "940305 000000" would list the new groups or news articles created since March 5, 1994.

The *group* parameter is only required when retrieving new news articles and should be set to NULL when retrieving new news groups. This parameter specifies from which news group the new news article listing is to be obtained.

The news groups or articles are delivered to the application by calling the application supplied callback function *proc* for each group or article. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, line, len, lParam)

GLOBALHANDLE hConn;
char far *line;
int len;
DWORD lParam;
```

An application can abort the listing by returning -1 in the callback function (all other return values are currently ignored) or by calling the **nntp\_xquit** function. Note that aborting a listing will also abort the underlying connection by resetting it. If the listing was aborted successfully, then the **nntp\_new** call will return NNTP\_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **nntp\_quit**) can be made until a new connection has been established with **nntp\_(x)open**.

Once the news groups or articles have been transferred, one final call to *proc* is made with the *len* parameter set to zero to indicate the end of the callbacks.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

The **nntp\_new** function is normally only used by NNTP servers.

**Return Value** The **nntp\_new** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.10 nntp\_open ()

**Description** Open connection with NNTP server.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_open (host, conn, proc, lParam)
```

<b>LPSTR</b> <i>host</i> ;	Name of NNTP server
<b>GLOBALHANDLE far</b> * <i>conn</i> ;	Buffer for handle of NNTP structure
<b>FARPROC</b> <i>proc</i> ;	Application callback function
<b>DWORD</b> <i>lParam</i> ;	Application supplied parameter

**Remarks** The `nntp_open` function establishes a TCP/IP connection with the specified host. The *host* parameter can point either to a machine name (possibly including a domain name) or to an internet address in dotted decimal notation (e.g. "192.1.2.3").

After a successful connection, the handle of the connection control structure is placed into the buffer pointed to by *conn*. This handle must be used in all subsequent calls to the other NNTP functions for this connection. The Distinct NNTP-32 library supports multiple concurrent NNTP connections limited only by system resources.

During development it may be necessary to observe exactly what commands are sent to the NNTP server and what replies are generated. For this purpose, the application can specify the address of a debug callback function with *proc*. This function will receive a copy of all network traffic and must be defined as follows.

```
int WINAPI (*proc) (hConn, buf, len, lParam)

GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

The *proc* parameter must be set to NULL if the application does not want to receive debug messages.

The *lParam* parameter can be set to any value that has meaning to the application such as an index into an array or a pointer to a structure. This parameter is passed back to the application as an argument in each callback function and allows the application to pass connection specific data to the callback function.

**Return Value** The `nntp_open` function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_FAILURE

Remote error.

NNTP\_CANNOT\_INITIALIZE

Support libraries or network drivers could not be initialized.

NNTP\_CANNOT\_INIT\_WINSOCK

Windows Sockets transport library could not be initialized.

NNTP\_OUT\_OF\_MEMORY

Not enough memory to allocate connection control structure.

NNTP\_UNKNOWN\_HOST

Unable to resolve specified host name.

NNTP\_CANNOT\_ALLOC\_SOCKET

Windows Sockets library could not allocate a socket.

NNTP\_CANNOT\_BIND\_SOCKET

Windows Sockets library was not able to bind socket to local port.

NNTP\_HOST\_NOT\_RESPONDING

Unable to connect to NNTP port on specified host.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.11 nntp\_option ()

**Description** Set connection option.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_option (hConn, option, value)
```

<b>GLOBALHANDLE</b> <i>hConn</i> ;	Handle of NNTP structure from <b>nntp_(x)open</b>
<b>WORD</b> <i>option</i> ;	Option to set
<b>DWORD</b> <i>value</i> ;	Value of option

**Remarks** The **nntp\_option** function sets options on a per connection basis. The *option* parameter specifies which option to set and must be equal to one of the following constants.

OPT_YIELD_ON	Enable yielding.
OPT_YIELD_OFF	Disable yielding.
OPT_SET_PARAM	Change callback parameter.
OPT_GET_PARAM	Get current callback parameter.
OPT_DEBUG	Change debug callback.
OPT_TIMEOUT	Change timeout value.

When enabling yielding with the OPT\_YIELD\_ON option, the *value* parameter can be used to specify the address of an application supplied yielding callback function. This function will be called during lengthy operations and should yield control to Windows so that other applications can process their messages. The function must be defined as follows.

```
int WINAPI (*proc) (void)
```

If an application wants to use the default NNTP-32 library yielding function **nntp\_yield**, then the *value* parameter must be set to NULL. The default yielding function should be sufficient in most cases.

The *value* parameter is not used for the OPT\_YIELD\_OFF option and must be set to NULL.

The callback parameter supplied by the application in the **nntp\_(x)open** call can be changed with the OPT\_SET\_PARAM option. To do so, set *value* to the new callback parameter. The current value of the callback parameter can be obtained with the OPT\_GET\_PARAM option. In this case, *value* must point to the DWORD which is to receive the current callback parameter (i.e. *value* must be a DWORD far \*).

The OPT\_DEBUG option allows the application to change or disable the debug callback function. To change the debug callback function, set *value* to the address of the new callback function. Note that this function address must be obtained by calling **MakeProcInstance** specifying an exported function. To disable debug callbacks, set *value* to NULL. Refer to the **nntp\_(x)open** description for more details on debug callback functions.

The OPT\_TIMEOUT option allows the application to set a timeout value other than the default for an open connection. The *value* parameter must specify the new timeout value in seconds and must be in the range of 1 to 999.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The `nntp_option` function either returns `NNTP_SUCCESS` or one of the following errors.

`NNTP_INVALID_HANDLE`

Invalid connection handle was specified.

`NNTP_INVALID_OPTION`

Invalid connection option was specified.

## 2.12 nntp\_position ()

**Description** Set current article.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_position (hConn, pos, flags)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
```

```
unsigned long pos;       Number of article
```

```
int flags;              Type of access
```

**Remarks** The **nntp\_position** function sets the current news article in the currently selected news group. The current article is set either to the next article or to the previous article or to the article specified with the *pos* parameter. The *flags* parameter determines how the current news article is set and must specify one of the following three values.

POSITION_PREV	Go to previous article.
POSITION_NEXT	Go to next article.
POSITION_SET	Go to specified article.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_position** function either returns NNTP\_SUCCESS or one of the following errors.

```
NNTP_INVALID_HANDLE
```

Invalid connection handle was specified.

```
NNTP_FAILURE
```

Host was not able to execute command.

```
NNTP_CANNOT_SEND_COMMAND
```

Windows Sockets library was not able to accept outgoing data.

```
NNTP_TIMED_OUT
```

Time out occurred while waiting for a response from host.

```
NNTP_IN_PROGRESS
```

Another operation is already in progress.

```
NNTP_ABORTED
```

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.13 nntp\_post\_buf ()

**Description** Post news article.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_post_buf (hConn, buf, len)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
```

```
char far * buf;          Buffer to send data
```

```
int len;                 Length of data
```

**Remarks** The **nntp\_post\_buf** function posts a news article to the current NNTP server. The *buf* parameter should contain the data and the *len* should specify the length of *buf*. The application can call **nntp\_post\_buf** number of times to post the entire article. One last call should be made with the *len* parameter set to 0 to signify the end of data.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_post\_buf** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.14 nntp\_post ()

**Description** Post news article.

```
#include <windows.h>
#include <d32-nntp.h>
```

```
int WINAPI nntp_post (hConn, proc)
```

```
GLOBALHANDLE hConn;    Handle of NNTP structure from nntp_(x)open
FARPROC proc;          Application callback function
```

**Remarks** The **nntp\_post** function posts a news article to the current server. The **nntp\_post** function uploads an article to the server by calling the application supplied callback function *proc* requesting message data. The *proc* callback is called until it returns zero to indicate the end of the message. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

An application can abort the posting of an article by returning -1 in the callback function or by calling the **nntp\_xquit** function. Note that aborting a posting will also abort the underlying connection by resetting it. If the posting was aborted successfully, then the **nntp\_post** call will return NNTP\_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **nntp\_quit**) can be made until a new connection has been established with **nntp\_(x)open**.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

**Return Value** The **nntp\_post** function either returns NNTP\_SUCCESS or one of the following errors.

```
NNTP_INVALID_HANDLE
```

Invalid connection handle was specified.

```
NNTP_FAILURE
```

Host was not able to execute command.

```
NNTP_CANNOT_SEND_COMMAND
```

Windows Sockets library was not able to accept outgoing data.

```
NNTP_TIMED_OUT
```

Time out occurred while waiting for a response from host.

```
NNTP_IN_PROGRESS
```

Another operation is already in progress.

```
NNTP_ABORTED
```

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.15 nntp\_quit ()

**Description** Disconnect from NNTP server.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_quit (hConn)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
```

**Remarks** The **nntp\_quit** function terminates an NNTP connection and frees all associated resources. The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open** and will be invalid once the **nntp\_quit** function returns.

**Return Value** The **nntp\_quit** function either returns NNTP\_SUCCESS or one of the following errors.

```
NNTP_INVALID_HANDLE
```

Invalid connection handle was specified.

```
NNTP_IN_PROGRESS
```

Another operation is already in progress.

## 2.16 nntp\_reply ()

**Description** Get last reply of NNTP server.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_reply (hConn, reply, len, code)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
```

```
char far *reply;         Buffer for last reply
```

```
int len;                 Size of reply buffer
```

```
int far *code;           Buffer for last reply code
```

**Remarks** The `nntp_reply` function retrieves the last response sent by the NNTP server. This can be useful in determining the cause of an `NNTP_FAILURE` error return. The buffer pointed to by `reply` will receive the entire response string from the server and the buffer pointed to by `code` will receive the integer equivalent of the reply code contained in the response.

The connection handle `hConn` must have been obtained with a call to `nntp_(x)open`.

**Return Value** The `nntp_reply` function either returns `NNTP_SUCCESS` or `NNTP_INVALID_HANDLE` to indicate that an invalid connection handle was specified.

## 2.17 nntp\_slave ()

**Description** Inform server of slave status.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_slave (hConn)
```

```
GLOBALHANDLE hConn;      Handle of NNTP structure from nntp_(x)open
```

**Remarks** The **nntp\_slave** function informs the current NNTP server that the local machine is acting as an intermediate or "slave" NNTP server. This information is only sent to advise the NNTP server and may be ignored by the server.

The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open**.

The **nntp\_slave** function is normally only used by NNTP servers.

**Return Value** The **nntp\_slave** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

Host was not able to execute command.

NNTP\_CANNOT\_SEND\_COMMAND

Windows Sockets library was not able to accept outgoing data.

NNTP\_TIMED\_OUT

Time out occurred while waiting for a response from host.

NNTP\_IN\_PROGRESS

Another operation is already in progress.

NNTP\_ABORTED

Operation was aborted and connection has been closed.

If this function returns NNTP\_FAILURE, then the **nntp\_reply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

## 2.18 nntp\_xopen ()

**Description** Open asynchronous connection with NNTP server.

```
#include <windows.h>
#include <d32-nntp.h>

int WINAPI nntp_xopen (host, conn, dbg, stat, lParam)

LPSTR host;           Name of NNTP server
GLOBALHANDLE far *conn; Buffer for handle of NNTP structure
FARPROC dbg;         Application debug callback function
FARPROC stat;        Application status callback function
DWORD lParam;        Application supplied parameter
```

**Remarks** The **nntp\_xopen** function establishes a TCP/IP connection with the specified host in an asynchronous manner. The *host* parameter can point either to a machine name (possibly including a domain name) or to an internet address in dotted decimal notation (e.g. "192.1.2.3").

Unlike the **nntp\_open** function, the **nntp\_xopen** function does not actually establish the connection before returning to the application. Instead, it allocates required local resources and returns the handle of the connection control structure in the buffer pointed to by *conn*. Once the connection has been established, the status callback function specified with *stat* will be called. This function must be defined as follows.

```
int WINAPI (*stat) (hConn, result, lParam)

GLOBALHANDLE hConn;
int result;
DWORD lParam;
```

If the connection was established successfully, then the *result* parameter will contain the value NNTP\_SUCCESS. If the connection could not be established, then the *result* parameter will contain one of the following errors.

```
NNTP_FAILURE
    Remote error.

>NNTP_UNKNOWN_HOST
    Unable to resolve specified host name.

>NNTP_CANNOT_ALLOC_SOCKET
    Windows Sockets library could not allocate a socket.

>NNTP_CANNOT_BIND_SOCKET
    Windows Sockets library was not able to bind socket to local port.

>NNTP_HOST_NOT_RESPONDING
    Unable to connect to NNTP port on specified host.

>NNTP_CANNOT_SEND_COMMAND
    Windows Sockets library was not able to accept outgoing data.

>NNTP_TIMED_OUT
    Time out occurred while waiting for a response from host.

>NNTP_ABORTED
    Asynchronous connection attempt aborted with call to nntp_xquit.
```

Once the status callback function has been called with *result* set to NNTP\_SUCCESS, then the handle placed into the buffer pointed to by *conn* must be used in all subsequent calls to the other NNTP functions for this connection. Any calls made with this handle before the connection has been established will fail with a return value of NNTP\_IN\_PROGRESS.

Multiple NNTP connections can be established simultaneously with the **nntp\_xopen** or **nntp\_open** calls. The Distinct NNTP-32 library supports multiple concurrent NNTP connections limited only by system resources.

During development it may be necessary to observe exactly what commands are sent to the NNTP server and what replies are generated. For this purpose, the application can specify the address of a debug callback function with *dbg*. This function will receive a copy of all network traffic and must be defined as follows.

```
int WINAPI (*dbg) (hConn, buf, len, lParam)

GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

The *dbg* parameter must be set to NULL if the application does not want to receive debug messages.

The *lParam* parameter can be set to any value that has meaning to the application such as an index into an array or a pointer to a structure. This parameter is passed back to the application as an argument in each callback function and allows the application to pass connection specific data to the callback function.

**Return Value** The **nntp\_xopen** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_CANNOT\_INITIALIZE

Support libraries or network drivers could not be initialized.

NNTP\_CANNOT\_INIT\_WINSOCK

Windows Sockets transport library could not be initialized.

NNTP\_OUT\_OF\_MEMORY

Not enough memory to allocate connection control structure.

## 2.19 nntp\_xquit ()

**Description** Cancel asynchronous connect or operation in progress.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_xquit (hConn)
```

**GLOBALHANDLE** *hConn*; Handle of NNTP structure from **nntp\_(x)open**

**Remarks** The **nntp\_xquit** function terminates an asynchronous connection attempt initiated with a call to **nntp\_xopen** or an NNTP operation in progress and frees all associated resources. The connection handle *hConn* must have been obtained with a call to **nntp\_(x)open** and will be invalid once the **nntp\_xquit** function returns.

If **nntp\_xquit** is called to cancel an asynchronous connect, then after the **nntp\_xquit** call returns, the application must not terminate until the status callback function *stat* specified in the **nntp\_xopen** call has been called with the *result* parameter set to NNTP\_ABORTED. If **nntp\_xquit** is called to cancel an operation in progress, then after the **nntp\_xquit** call returns, the application must not terminate until the canceled function returns.

The **nntp\_xquit** function can be called after calling **nntp\_xopen** and before the status callback function *stat* specified in the **nntp\_xopen** call is called with the result of the connection attempt. The **nntp\_xquit** function can also be called to cancel an **nntp\_article**, **nntp\_forward**, **nntp\_group**, **nntp\_help**, **nntp\_list**, **nntp\_new**, **nntp\_position**, **nntp\_post** and **nntp\_slave** operation in progress.

**Return Value** The **nntp\_xquit** function either returns NNTP\_SUCCESS or one of the following errors.

NNTP\_INVALID\_HANDLE

Invalid connection handle was specified.

NNTP\_FAILURE

No asynchronous connect in progress or asynchronous connect already aborted.

## 2.20 nntp\_yield ()

**Description** Yield control to Windows.

```
#include <windows.h>
```

```
#include <d32-nntp.h>
```

```
int WINAPI nntp_yield (void)
```

**Remarks** The **nntp\_yield** function yields control to Windows so that other applications can process their messages. Every time this function is called, all queued messages for all applications (including the application calling the NNTP-32 library) are removed from the system queue and delivered to the applications for processing.

This function is exported in case the calling application needs to yield control to Windows, for example while processing data in a callback function.

**Return Value** The **nntp\_yield** function returns the number of messages that were processed.