21.11. `nntplib` — NNTP protocol client¶

This module defines the class `NNTP` which implements the client side of the NNTP protocol. It can be used to implement a news reader or poster, or automated news processors. For more information on NNTP (Network News Transfer Protocol), see Internet **RFC 977**.

Here are two small examples of how it can be used. To list some statistics about a newsgroup and print the subjects of the last 10 articles:

```
>>> s = NNTP('news.cwi.nl')
>>> resp, count, first, last, name = s.group('comp.lang.python')
>>> print 'Group', name, 'has', count, 'articles, range', first, 'to', last
Group comp.lang.python has 59 articles, range 3742 to 3803
>>> resp, subs = s.xhdr('subject', first + '-' + last)
>>> for id, sub in subs[-10:]: print id, sub
...
3792 Re: Removing elements from a list while iterating...
3793 Re: Who likes Info files?
3794 Emacs and doc strings
3795 a few questions about the Mac implementation
3796 Re: executable python scripts
3797 Re: executable python scripts
3798 Re: a few questions about the Mac implementation
3799 Re: PROPOSAL: A Generic Python Object Interface for Python C Modules
3802 Re: executable python scripts
3803 Re: \POSIX{} wait and SIGCHLD
>>> s.quit()
'205 news.cwi.nl closing connection.  Goodbye.'
```

To post an article from a file (this assumes that the article has valid headers):

```
>>> s = NNTP('news.cwi.nl')
>>> f = open('/tmp/article')
>>> s.post(f)
'240 Article posted successfully.'
>>> s.quit()
'205 news.cwi.nl closing connection.  Goodbye.'
```

The module itself defines the following items:

class `nntplib.NNTP`(*host*[, *port*[, *user*[, *password*[, *readermode*][, *usenetrc*]]]])¶

Return a new instance of the `NNTP` class, representing a connection to the NNTP server running on host *host*, listening at port *port*. The default *port* is 119. If the optional *user* and *password* are provided, or if suitable credentials are present in `/.netrc` and the optional flag *usenetrc* is true (the default), the `AUTHINFO USER` and `AUTHINFO PASS` commands are used to identify and authenticate the user to the server. If the optional flag *readermode* is true, then a `mode reader` command is sent before authentication is performed. Reader mode is sometimes necessary if you are connecting to an NNTP server on the local machine and intend to call reader-specific commands, such as `group`. If you get unexpected `NNTPPermanentError`s, you might need to set *readermode*. *readermode* defaults to `None`. *usenetrc* defaults to `True`.

Changed in version 2.4: *usenetrc* argument added.

exception `nntplib.NNTPError`¶

Derived from the standard exception `Exception`, this is the base class for all exceptions raised by the `nntplib` module.

exception `nntplib.NNTPReplyError`¶

Exception raised when an unexpected reply is received from the server. For backwards compatibility, the exception `error_reply` is equivalent to this class.

exception `nntplib.NNTPTemporaryError`¶

Exception raised when an error code in the range 400–499 is received. For backwards compatibility, the exception `error_temp` is equivalent to this class.

exception `nntplib.NNTPPermanentError`¶

Exception raised when an error code in the range 500–599 is received. For backwards compatibility, the exception `error_perm` is equivalent to this class.

exception `nntplib.NNTPProtocolError`¶

Exception raised when a reply is received from the server that does not begin with a digit in the range 1–5. For backwards compatibility, the exception `error_proto` is equivalent to this class.

exception `nntplib.NNTPDataError`¶

Exception raised when there is some error in the response data. For backwards compatibility, the exception `error_data` is equivalent to this class.

21.11.1. NNTP Objects¶

NNTP instances have the following methods. The *response* that is returned as the first item in the return tuple of almost all methods is the server's response: a string beginning with a three-digit code. If the server's response indicates an error, the method raises one of the above exceptions.

`NNTP.getwelcome`()¶

Return the welcome message sent by the server in reply to the initial connection. (This message sometimes contains disclaimers or help information that may be relevant to the user.)

`NNTP.set_debuglevel`(*level*)¶

Set the instance's debugging level. This controls the amount of debugging output printed. The default, `0`, produces no debugging output. A value of `1` produces a moderate amount of debugging output, generally a single line per request or response. A value of 2 or higher produces the maximum amount of debugging output, logging each line sent and received on the connection (including message text).

`NNTP.newgroups`(*date*, *time*[, *file*])¶

Send a `NEWGROUPS` command. The *date* argument should be a string of the form `'yymmdd'` indicating the date, and *time* should be a string of the form `'hhmmss'` indicating the time. Return a pair `(response, groups)` where *groups* is a list of group names that are new since the given date and time. If the *file* parameter is supplied, then the output of the `NEWGROUPS` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list.

`NNTP.newnews`(*group*, *date*, *time*[, *file*])¶

Send a `NEWNEWS` command. Here, *group* is a group name or `'*'`, and *date* and *time* have the same meaning as for `newgroups()`. Return a pair `(response, articles)` where *articles* is a list of message ids. If the *file* parameter is supplied, then the output of the `NEWNEWS` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list.

`NNTP.list`([*file*])¶

Send a `LIST` command. Return a pair `(response, list)` where *list* is a list of tuples. Each tuple has the form `(group, last, first, flag)`, where *group* is a group name, *last* and *first* are the last and first article numbers (as strings), and *flag* is `'y'` if posting is allowed, `'n'` if not, and `'m'` if the newsgroup is moderated. (Note the ordering: *last*, *first*.) If the *file* parameter is supplied, then the output of the `LIST` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list.

`NNTP.descriptions`(*grouppattern*)¶

Send a `LIST NEWSGROUPS` command, where *grouppattern* is a wildmat string as specified in RFC2980 (it's essentially the same as DOS or UNIX shell wildcard strings). Return a pair `(response, list)`, where *list* is a list of tuples containing `(name, title)`.

New in version 2.4.

`NNTP.description`(*group*)¶

Get a description for a single group *group*. If more than one group matches (if 'group' is a real wildmat string), return the first match. If no group matches, return an empty string.

This elides the response code from the server. If the response code is needed, use `descriptions()`.

New in version 2.4.

`NNTP.group`(*name*)¶

Send a `GROUP` command, where *name* is the group name. Return a tuple `(response, count, first, last, name)` where *count* is the (estimated) number of articles in the group, *first* is the first article number in the group, *last* is the last article number in the group, and *name* is the group name. The numbers are returned as strings.

`NNTP.help`([*file*])¶

Send a `HELP` command. Return a pair `(response, list)` where *list* is a list of help strings. If the *file* parameter is supplied, then the output of the `HELP` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list.

`NNTP.stat`(*id*)¶

Send a `STAT` command, where *id* is the message id (enclosed in `'<'` and `'>'`) or an article number (as a string). Return a triple `(response, number, id)` where *number* is the article number (as a string) and *id* is the message id (enclosed in `'<'` and `'>'`).

`NNTP.next`()¶

Send a `NEXT` command. Return as for `stat()`.

`NNTP.last`()¶

Send a `LAST` command. Return as for `stat()`.

`NNTP.head`(*id*)¶
Send a `HEAD` command, where *id* has the same meaning as for `stat()`. Return a tuple `(response, number, id, list)` where the first three are the same as for `stat()`, and *list* is a list of the article's headers (an uninterpreted list of lines, without trailing newlines).

`NNTP.body`(*id*[, *file*])¶
Send a `BODY` command, where *id* has the same meaning as for `stat()`. If the *file* parameter is supplied, then the body is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the body. Return as for `head()`. If *file* is supplied, then the returned *list* is an empty list.

`NNTP.article`(*id*)¶
Send an `ARTICLE` command, where *id* has the same meaning as for `stat()`. Return as for `head()`.

`NNTP.slave`()¶
Send a `SLAVE` command. Return the server's *response*.

`NNTP.xhdr`(*header*, *string*[, *file*])¶
Send an `XHDR` command. This command is not defined in the RFC but is a common extension. The *header* argument is a header keyword, e.g. `'subject'`. The *string* argument should have the form `'first-last'` where *first* and *last* are the first and last article numbers to search. Return a pair `(response, list)`, where *list* is a list of pairs `(id, text)`, where *id* is an article number (as a string) and *text* is the text of the requested header for that article. If the *file* parameter is supplied, then the output of the `XHDR` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list.

`NNTP.post`(*file*)¶
Post an article using the `POST` command. The *file* argument is an open file object which is read until EOF using its `readline()` method. It should be a well-formed news article, including the required headers. The `post()` method automatically escapes lines beginning with `..`

`NNTP.ihave`(*id*, *file*)¶
Send an `IHAVE` command. *id* is a message id (enclosed in `'<'` and `'>'`). If the response is not an error, treat *file* exactly as for the `post()` method.

`NNTP.date`()¶
Return a triple `(response, date, time)`, containing the current date and time in a form suitable for the `newnews()` and `newgroups()` methods. This is an optional NNTP extension, and may not be supported by all servers.

`NNTP.xgtitle`(*name*[, *file*])¶

Process an `XGTITLE` command, returning a pair `(response, list)`, where *list* is a list of tuples containing `(name, title)`. If the *file* parameter is supplied, then the output of the `XGTITLE` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list. This is an optional NNTP extension, and may not be supported by all servers.

RFC2980 says "It is suggested that this extension be deprecated". Use `descriptions()` or `description()` instead.

`NNTP.xover`(*start*, *end*[, *file*])¶
Return a pair `(resp, list)`. *list* is a list of tuples, one for each article in the range delimited by the *start* and *end* article numbers. Each tuple is of the form `(article number, subject, poster, date, id, references, size, lines)`. If the *file* parameter is supplied, then the output of the `XOVER` command is stored in a file. If *file* is a string, then the method will open a file object with that name, write to it then close it. If *file* is a file object, then it will start calling `write()` on it to store the lines of the command output. If *file* is supplied, then the returned *list* is an empty list. This is an optional NNTP extension, and may not be supported by all servers.

`NNTP.xpath`(*id*)¶
Return a pair `(resp, path)`, where *path* is the directory path to the article with message ID *id*. This is an optional NNTP extension, and may not be supported by all servers.

`NNTP.quit`()¶
Send a `QUIT` command and close the connection. Once this method has been called, no other methods of the NNTP object should be called.

Last updated on Feb 26, 2010. Created using Sphinx 0.6.3.