`!ftplib` --- FTP protocol client

- Reference
  - FTP objects
  - FTP_TLS objects
  - Module variables

ftplib

Source code: `Lib/ftplib.py`

pair: FTP; protocol single: FTP; ftplib (standard module)

This module defines the class `FTP` and a few related items. The `FTP` class implements the client side of the FTP protocol. You can use this to write Python programs that perform a variety of automated FTP jobs, such as mirroring other FTP servers. It is also used by the module `urllib.request` to handle URLs that use FTP. For more information on FTP (File Transfer Protocol), see internet `959`.

The default encoding is UTF-8, following `2640`.

not WASI.

This module does not work or is not available on WebAssembly. See `wasm-availability` for more information.

Here's a sample session using the `!ftplib` module:

```
>>> from ftplib import FTP
>>> ftp = FTP('ftp.us.debian.org')  # connect to host, default port
>>> ftp.login()                     # user anonymous, passwd anonymous@
'230 Login successful.'
```

```
>>> ftp.cwd('debian')               # change into "debian" directory
'250 Directory successfully changed.'
>>> ftp.retrlines('LIST')           # list directory contents
-rw-rw-r--    1 1176     1176         1063 Jun 15 10:18 README
...
drwxr-sr-x    5 1176     1176         4096 Dec 19  2000 pool
drwxr-sr-x    4 1176     1176         4096 Nov 17  2008 project
drwxr-xr-x    3 1176     1176         4096 Oct 10  2012 tools
'226 Directory send OK.'
>>> with open('README', 'wb') as fp:
...     ftp.retrbinary('RETR README', fp.write)
'226 Transfer complete.'
>>> ftp.quit()
'221 Goodbye.'
```

Reference

FTP objects

Return a new instance of the `FTP` class.

param str host
The hostname to connect to. If given, `connect(host)` is implicitly called by the constructor.

param str user
The username to log in with (default: `'anonymous'`). If given, `login(host, passwd, acct)` is implicitly called by the constructor.

param str passwd
The password to use when logging in. If not given, and if *passwd* is the empty string or `"-"`, a password will be automatically generated.

param str acct

Account information to be used for the `ACCT` FTP command. Few systems implement this. See RFC-959 for more details.

param timeout
A timeout in seconds for blocking operations like `connect` (default: the global default timeout

setting).

type timeout
float | None

param source_address
A 2-tuple `(host, port)` for the socket to bind to as its source address before connecting.

type source_address
tuple | None

param str encoding
The encoding for directories and filenames (default: `'utf-8'`).

The `FTP` class supports the `with` statement, e.g.:

> >>> from ftplib import FTP >>> with FTP("ftp1.at.proftpd.org") as ftp: ... ftp.login() ... ftp.dir() ... # doctest: +SKIP '230 Anonymous login ok, restrictions apply.' dr-xr-xr-x 9 ftp ftp 154 May 6 10:43 . dr-xr-xr-x 9 ftp ftp 154 May 6 10:43 .. dr-xr-xr-x 5 ftp ftp 4096 May 6 10:43 CentOS dr-xr-xr-x 3 ftp ftp 18 Jul 10 2008 Fedora >>>

3.2 Support for the `with` statement was added.

3.3 *source_address* parameter was added.

3.9 If the *timeout* parameter is set to be zero, it will raise a `ValueError` to prevent the creation of a non-blocking socket. The *encoding* parameter was added, and the default was changed from Latin-1 to UTF-8 to follow `2640`.

Several `!FTP` methods are available in two flavors: one for handling text files and another for binary files. The methods are named for the command which is used followed by `lines` for the text version or `binary` for the binary version.

`FTP` instances have the following methods:

FTP.set_debuglevel(level)

Set the instance's debugging level as an `int`. This controls the amount of debugging output printed. The debug levels are:

- `0` (default): No debug output.
- `1`: Produce a moderate amount of debug output, generally a single line per request.

- `2` or higher: Produce the maximum amount of debugging output, logging each line sent and received on the control connection.

FTP.connect(host='', port=0, timeout=None, source_address=None)

Connect to the given host and port. This function should be called only once for each instance; it should not be called if a *host* argument was given when the `FTP` instance was created. All other `!FTP` methods can only be called after a connection has successfully been made.

param str host
The host to connect to.

param int port
The TCP port to connect to (default: `21`, as specified by the FTP protocol specification). It is rarely needed to specify a different port number.

param timeout
A timeout in seconds for the connection attempt (default: the global default timeout setting).

type timeout
float | None

param source_address
A 2-tuple `(host, port)` for the socket to bind to as its source address before connecting.

type source_address
tuple | None

ftplib.connect self,host,port ftplib.FTP.connect

3.3 *source_address* parameter was added.

FTP.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.)

FTP.login(user='anonymous', passwd='', acct='')

Log on to the connected FTP server. This function should be called only once for each instance, after a connection has been established; it should not be called if the *host* and *user* arguments were given when the `FTP` instance was created. Most FTP commands are only allowed after the client has logged in.

param str user
The username to log in with (default: `'anonymous'`).

param str passwd
The password to use when logging in. If not given, and if *passwd* is the empty string or `"-"`, a password will be automatically generated.

param str acct

Account information to be used for the `ACCT` FTP command. Few systems implement this. See RFC-959 for more details.

FTP.abort()

Abort a file transfer that is in progress. Using this does not always work, but it's worth a try.

FTP.sendcmd(cmd)

Send a simple command string to the server and return the response string.

ftplib.sendcmd self,cmd ftplib.FTP.sendcmd

FTP.voidcmd(cmd)

Send a simple command string to the server and handle the response. Return the response string if the response code corresponds to success (codes in the range 200--299). Raise `error_reply` otherwise.

ftplib.sendcmd self,cmd ftplib.FTP.voidcmd

FTP.retrbinary(cmd, callback, blocksize=8192, rest=None)

Retrieve a file in binary transfer mode.

param str cmd
An appropriate `RETR` command: `"RETR {filename}"`.

param callback
A single parameter callable that is called for each block of data received, with its single argument being the data as `bytes`.

type callback
`callable`

param int blocksize
The maximum chunk size to read on the low-level `~socket.socket` object created to do the actual transfer. This also corresponds to the largest size of data that will be passed to *callback*. Defaults to `8192`.

param int rest
A `REST` command to be sent to the server. See the documentation for the *rest* parameter of the `transfercmd` method.

FTP.retrlines(cmd, callback=None)

Retrieve a file or directory listing in the encoding specified by the *encoding* parameter at initialization. *cmd* should be an appropriate `RETR` command (see `retrbinary`) or a command such as `LIST` or `NLST` (usually just the string `'LIST'`). `LIST` retrieves a list of files and information about those files. `NLST` retrieves a list of file names. The *callback* function is called for each line with a string argument containing the line with the trailing CRLF stripped. The default *callback* prints the line to `sys.stdout`.

FTP.set_pasv(val)

Enable "passive" mode if *val* is true, otherwise disable passive mode. Passive mode is on by default.

FTP.storbinary(cmd, fp, blocksize=8192, callback=None, rest=None)

Store a file in binary transfer mode.

param str cmd
An appropriate `STOR` command: `"STOR {filename}"`.

param fp
A file object (opened in binary mode) which is read until EOF, using its `~io.RawIOBase.read` method in blocks of size *blocksize* to provide the data to be stored.

type fp
`file object`

param int blocksize
The read block size. Defaults to `8192`.

param callback
A single parameter callable that is called for each block of data sent, with its single argument being the data as `bytes`.

type callback
`callable`

param int rest
A `REST` command to be sent to the server. See the documentation for the *rest* parameter of the `transfercmd` method.

3.2 The *rest* parameter was added.

FTP.storlines(cmd, fp, callback=None)

Store a file in line mode. *cmd* should be an appropriate `STOR` command (see `storbinary`). Lines are read until EOF from the `file object` *fp* (opened in binary mode) using its `~io.IOBase.readline` method to provide the data to be stored. *callback* is an optional single parameter callable that is called on each line after it is sent.

FTP.transfercmd(cmd, rest=None)

Initiate a transfer over the data connection. If the transfer is active, send an `EPRT` or `PORT` command and the transfer command specified by *cmd*, and accept the connection. If the server is passive, send an `EPSV` or `PASV` command, connect to it, and start the transfer command. Either way, return the socket for the connection.

If optional *rest* is given, a `REST` command is sent to the server, passing *rest* as an argument. *rest* is usually a byte offset into the requested file, telling the server to restart sending the file's bytes at the requested offset, skipping over the initial bytes. Note however that the `transfercmd` method converts *rest* to a string with the *encoding* parameter specified at initialization, but no check is performed on the string's contents. If the server does not recognize the `REST` command, an `error_reply` exception will be raised. If this happens, simply call `transfercmd` without a *rest* argument.

FTP.ntransfercmd(cmd, rest=None)

Like `transfercmd`, but returns a tuple of the data connection and the expected size of the data. If the expected size could not be computed, `None` will be returned as the expected size. *cmd* and *rest* means the same thing as in `transfercmd`.

FTP.mlsd(path="", facts=[])

List a directory in a standardized format by using `MLSD` command (`3659`). If *path* is omitted the current directory is assumed. *facts* is a list of strings representing the type of information desired (e.g. `["type", "size", "perm"]`). Return a generator object yielding a tuple of two elements for every file found in path. First element is the file name, the second one is a dictionary containing facts about the file name. Content of this dictionary might be limited by the *facts* argument but server is not guaranteed to return all requested facts.

3.3

FTP.nlst(argument[, ...])

Return a list of file names as returned by the `NLST` command. The optional *argument* is a directory to list (default is the current server directory). Multiple arguments can be used to pass non-standard options to the `NLST` command.

> [!NOTE] If your server supports the command, `mlsd` offers a better API.

FTP.dir(argument[, ...])

Produce a directory listing as returned by the `LIST` command, printing it to standard output. The optional *argument* is a directory to list (default is the current server directory). Multiple arguments can be used to pass non-standard options to the `LIST` command. If the last argument is a function, it is used as a *callback* function as for `retrlines`; the default prints to `sys.stdout`. This method returns `None`.

> [!NOTE] If your server supports the command, `mlsd` offers a better API.

FTP.rename(fromname, toname)

Rename file *fromname* on the server to *toname*.

FTP.delete(filename)

Remove the file named *filename* from the server. If successful, returns the text of the response, otherwise raises `error_perm` on permission errors or `error_reply` on other errors.

FTP.cwd(pathname)

Set the current directory on the server.

FTP.mkd(pathname)

Create a new directory on the server.

FTP.pwd()

Return the pathname of the current directory on the server.

FTP.rmd(dirname)

Remove the directory named *dirname* on the server.

FTP.size(filename)

Request the size of the file named *filename* on the server. On success, the size of the file is returned as an integer, otherwise `None` is returned. Note that the `SIZE` command is not standardized, but is supported by many common server implementations.

FTP.quit()

Send a `QUIT` command to the server and close the connection. This is the "polite" way to close a connection, but it may raise an exception if the server responds with an error to the `QUIT` command. This implies a call to the `close` method which renders the `FTP` instance useless for subsequent calls (see below).

FTP.close()

Close the connection unilaterally. This should not be applied to an already closed connection such as after a successful call to `~FTP.quit`. After this call the `FTP` instance should not be used any more (after a call to `close` or `~FTP.quit` you cannot reopen the connection by issuing another `login` method).

FTP_TLS objects

An `FTP` subclass which adds TLS support to FTP as described in `4217`. Connect to port 21 implicitly securing the FTP control connection before authenticating.

> [!NOTE] The user must explicitly secure the data connection by calling the `prot_p` method.

param str host
The hostname to connect to. If given, `connect(host)` is implicitly called by the constructor.

param str user
The username to log in with (default: `'anonymous'`). If given, `login(host, passwd, acct)` is implicitly called by the constructor.

param str passwd
The password to use when logging in. If not given, and if *passwd* is the empty string or `"-"`, a

password will be automatically generated.

param str acct

Account information to be used for the `ACCT` FTP command. Few systems implement this. See RFC-959 for more details.

param context
An SSL context object which allows bundling SSL configuration options, certificates and private keys into a single, potentially long-lived, structure. Please read `ssl-security` for best practices.

type context
`ssl.SSLContext`

param timeout
A timeout in seconds for blocking operations like `~FTP.connect` (default: the global default timeout setting).

type timeout
float | None

param source_address
A 2-tuple `(host, port)` for the socket to bind to as its source address before connecting.

type source_address
tuple | None

param str encoding
The encoding for directories and filenames (default: `'utf-8'`).

3.2

3.3 Added the *source_address* parameter.

3.4 The class now supports hostname check with `ssl.SSLContext.check_hostname` and *Server Name Indication* (see `ssl.HAS_SNI`).

3.9 If the *timeout* parameter is set to be zero, it will raise a `ValueError` to prevent the creation of a non-blocking socket. The *encoding* parameter was added, and the default was changed from Latin-1 to UTF-8 to follow `2640`.

3.12 The deprecated *keyfile* and *certfile* parameters have been removed.

Here's a sample session using the `FTP_TLS` class:

```
>>> ftps = FTP_TLS('ftp.pureftpd.org')
>>> ftps.login()
'230 Anonymous user logged in'
>>> ftps.prot_p()
'200 Data protection level set to "private"'
>>> ftps.nlst()
['6jack', 'OpenBSD', 'antilink', 'blogbench', 'bsdcam', 'clockspeed', 'djbdns-jedi', 'docs',
'eaccelerator-jedi', 'favicon.ico', 'francotone', 'fugu', 'ignore', 'libpuzzle', 'metalog',
'minidentd', 'misc', 'mysql-udf-global-user-variables', 'php-jenkins-hash', 'php-skein-hash',
'php-webdav', 'phpaudit', 'phpbench', 'pincaster', 'ping', 'posto', 'pub', 'public',
'public_keys', 'pure-ftpd', 'qscan', 'qtc', 'sharedance', 'skycache', 'sound', 'tmp', 'ucarp']
```

`!FTP_TLS` class inherits from `FTP`, defining these additional methods and attributes:

FTP_TLS.auth()

Set up a secure control connection by using TLS.

3.4 The method now supports hostname check with `ssl.SSLContext.check_hostname` and *Server Name Indication* (see `ssl.HAS_SNI`).

FTP_TLS.ccc()

Revert control channel back to plaintext. This can be useful to take advantage of firewalls that know how to handle NAT with non-secure FTP without opening fixed ports.

3.3

FTP_TLS.prot_p()

Set up secure data connection by using TLS.

FTP_TLS.prot_c()

Set up clear text data connection.

Module variables

error_reply

Exception raised when an unexpected reply is received from the server.

error_temp

Exception raised when an error code signifying a temporary error (response codes in the range 400--499) is received.

error_perm

Exception raised when an error code signifying a permanent error (response codes in the range 500--599) is received.

error_proto

Exception raised when a reply is received from the server that does not fit the response specifications of the File Transfer Protocol, i.e. begin with a digit in the range 1--5.

all_errors

The set of all exceptions (as a tuple) that methods of `FTP` instances may raise as a result of problems with the FTP connection (as opposed to programming errors made by the caller). This set includes the four exceptions listed above as well as `OSError` and `EOFError`.

Module `netrc`
Parser for the `.netrc` file format. The file `.netrc` is typically used by FTP clients to load user authentication information before prompting the user.

□□□ #2
□ □□□ □□□ 2026-10-31 17:10:34 UTC
□ □□□ □□□ 2026-10-31 17:10:39 UTC