poplib — POP3 protocol client

Source code: Lib/poplib.py

This module defines a class, POP3, which encapsulates a connection to a POP3 server and implements the protocol as defined in **RFC 1939**. The POP3 class supports both the minimal and optional command sets from **RFC 1939**. The POP3 class also supports the STLS command introduced in **RFC 2595** to enable encrypted communication on an already established connection.

Additionally, this module provides a class POP3_SSL, which provides support for connecting to POP3 servers that use SSL as an underlying protocol layer.

Note that POP3, though widely supported, is obsolescent. The implementation quality of POP3 servers varies widely, and too many are quite poor. If your mailserver supports IMAP, you would be better off using the imaplib.IMAP4 class, as IMAP servers tend to be better implemented.

The poplib module provides two classes:

class poplib.**POP3**(*host*, *port=POP3_PORT*[, *timeout*])

This class implements the actual POP3 protocol. The connection is created when the instance is initialized. If *port* is omitted, the standard POP3 port (110) is used. The optional *timeout* parameter specifies a timeout in seconds for the connection attempt (if not specified, the global default timeout setting will be used).

Raises an auditing event poplib.connect with arguments self, host, port.

All commands will raise an auditing event poplib.putline with arguments self and line, where line is the bytes about to be sent to the remote host.

Changed in version 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.

class poplib.**POP3_SSL**(*host*, *port=POP3_SSL_PORT*, *keyfile=None*, *certfile=None*, *timeout=None*, *context=None*)

This is a subclass of POP3 that connects to the server over an SSL encrypted socket. If *port* is not specified, 995, the standard POP3-over-SSL port is used. *timeout* works as in the POP3 constructor. *context* is an optional ssl.SSLContext object which allows bundling SSL configuration options, certificates and private keys into a single (potentially long-lived) structure. Please read Security considerations for best practices.

keyfile and *certfile* are a legacy alternative to *context* - they can point to PEM-formatted private key and certificate chain files, respectively, for the SSL connection.

Raises an auditing event poplib.connect with arguments self, host, port.

All commands will raise an auditing event `poplib.putline` with arguments `self` and `line`, where `line` is the bytes about to be sent to the remote host.

Changed in version 3.2: context parameter added.

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

Deprecated since version 3.6: keyfile and *certfile* are deprecated in favor of *context*. Please use `ssl.SSLContext.load_cert_chain()` instead, or let `ssl.create_default_context()` select the system's trusted CA certificates for you.

Changed in version 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.

One exception is defined as an attribute of the `poplib` module:

exception `poplib.`**`error_proto`**

Exception raised on any errors from this module (errors from `socket` module are not caught). The reason for the exception is passed to the constructor as a string.

See also:

Module `imaplib`

The standard Python IMAP module.

Frequently Asked Questions About Fetchmail

The FAQ for the **fetchmail** POP/IMAP client collects information on POP3 server variations and RFC noncompliance that may be useful if you need to write an application based on the POP protocol.

POP3 Objects

All POP3 commands are represented by methods of the same name, in lower-case; most return the response text sent by the server.

An `POP3` instance has the following methods:

`POP3.`**`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. A value of `2` or higher produces the maximum amount of debugging output, logging each line sent and received on the control connection.

`POP3.`**`getwelcome`**`()`

Returns the greeting string sent by the POP3 server.

POP3.**capa**()

Query the server's capabilities as specified in **RFC 2449**. Returns a dictionary in the form `{'name': ['param'...]}`.

New in version 3.4.

POP3.**user**(*username*)

Send user command, response should indicate that a password is required.

POP3.**pass_**(*password*)

Send password, response includes message count and mailbox size. Note: the mailbox on the server is locked until `quit()` is called.

POP3.**apop**(*user*, *secret*)

Use the more secure APOP authentication to log into the POP3 server.

POP3.**rpop**(*user*)

Use RPOP authentication (similar to UNIX r-commands) to log into POP3 server.

POP3.**stat**()

Get mailbox status. The result is a tuple of 2 integers: `(message count, mailbox size)`.

POP3.**list**([*which*])

Request message list, result is in the form `(response, ['mesg_num octets', ...], octets)`. If *which* is set, it is the message to list.

POP3.**retr**(*which*)

Retrieve whole message number *which*, and set its seen flag. Result is in form `(response, ['line', ...], octets)`.

POP3.**dele**(*which*)

Flag message number *which* for deletion. On most servers deletions are not actually performed until QUIT (the major exception is Eudora QPOP, which deliberately violates the RFCs by doing pending deletes on any disconnect).

POP3.**rset**()

Remove any deletion marks for the mailbox.

POP3.**noop**()

Do nothing. Might be used as a keep-alive.

POP3.**quit**()

Signoff: commit changes, unlock mailbox, drop connection.

POP3.**top**(*which*, *howmuch*)

Retrieves the message header plus *howmuch* lines of the message after the header of message number *which*. Result is in form `(response, ['line', ...], octets)`.

The POP3 TOP command this method uses, unlike the RETR command, doesn't set the message's seen flag; unfortunately, TOP is poorly specified in the RFCs and is frequently broken in off-brand servers. Test this method by hand against the POP3 servers you will use before trusting it.

`POP3.`**`uidl`**(*`which=None`*)

Return message digest (unique id) list. If *which* is specified, result contains the unique id for that message in the form `'response mesgnum uid`, otherwise result is list `(response, ['mesgnum uid', ...], octets)`.

`POP3.`**`utf8`**()

Try to switch to UTF-8 mode. Returns the server response if successful, raises `error_proto` if not. Specified in **RFC 6856**.

New in version 3.5.

`POP3.`**`stls`**(*`context=None`*)

Start a TLS session on the active connection as specified in **RFC 2595**. This is only allowed before user authentication

context parameter is a `ssl.SSLContext` object which allows bundling SSL configuration options, certificates and private keys into a single (potentially long-lived) structure. Please read Security considerations for best practices.

This method supports hostname checking via `ssl.SSLContext.check_hostname` and *Server Name Indication* (see `ssl.HAS_SNI`).

New in version 3.4.

Instances of `POP3_SSL` have no additional methods. The interface of this subclass is identical to its parent.

POP3 Example

Here is a minimal example (without error checking) that opens a mailbox and retrieves and prints all messages:

```
import getpass, poplib

M = poplib.POP3('localhost')
M.user(getpass.getuser())
M.pass_(getpass.getpass())
numMessages = len(M.list()[1])
for i in range(numMessages):
    for j in M.retr(i+1)[1]:
        print(j)
```

At the end of the module, there is a test section that contains a more extensive example of usage.