imaplib — IMAP4 protocol client

Source code: Lib/imaplib.py

This module defines three classes, `IMAP4`, `IMAP4_SSL` and `IMAP4_stream`, which encapsulate a connection to an IMAP4 server and implement a large subset of the IMAP4rev1 client protocol as defined in **RFC 2060**. It is backward compatible with IMAP4 (**RFC 1730**) servers, but note that the `STATUS` command is not supported in IMAP4.

Three classes are provided by the `imaplib` module, `IMAP4` is the base class:

class `imaplib.`**`IMAP4`**(*`host=''`*, *`port=IMAP4_PORT`*, *`timeout=None`*)

This class implements the actual IMAP4 protocol. The connection is created and protocol version (IMAP4 or IMAP4rev1) is determined when the instance is initialized. If *host* is not specified, `''` (the local host) is used. If *port* is omitted, the standard IMAP4 port (143) is used. The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If timeout is not given or is None, the global default socket timeout is used.

The `IMAP4` class supports the `with` statement. When used like this, the IMAP4 `LOGOUT` command is issued automatically when the `with` statement exits. E.g.:

```
>>> from imaplib import IMAP4
>>> with IMAP4("domain.org") as M:
...     M.noop()
...
('OK', [b'Nothing Accomplished. d25if65hy903weo.87'])
```

Changed in version 3.5: Support for the `with` statement was added.

Changed in version 3.9: The optional *timeout* parameter was added.

Three exceptions are defined as attributes of the `IMAP4` class:

exception `IMAP4.`**`error`**

Exception raised on any errors. The reason for the exception is passed to the constructor as a string.

exception `IMAP4.`**`abort`**

IMAP4 server errors cause this exception to be raised. This is a sub-class of `IMAP4.error`. Note that closing the instance and instantiating a new one will usually allow recovery from this exception.

exception `IMAP4.`**`readonly`**

This exception is raised when a writable mailbox has its status changed by the server. This is a sub-class of `IMAP4.error`. Some other client now has write permission, and the mailbox will need to be re-opened to re-obtain write permission.

There's also a subclass for secure connections:

class imaplib.**IMAP4_SSL**(*host=''*, *port=IMAP4_SSL_PORT*, *keyfile=None*, *certfile=None*, *ssl_context=None*, *timeout=None*)

This is a subclass derived from IMAP4 that connects over an SSL encrypted socket (to use this class you need a socket module that was compiled with SSL support). If *host* is not specified, '' (the local host) is used. If *port* is omitted, the standard IMAP4-over-SSL port (993) is used. *ssl_context* 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.

keyfile and *certfile* are a legacy alternative to *ssl_context* - they can point to PEM-formatted private key and certificate chain files for the SSL connection. Note that the *keyfile*/*certfile* parameters are mutually exclusive with *ssl_context*, a ValueError is raised if *keyfile*/*certfile* is provided along with *ssl_context*.

The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If timeout is not given or is None, the global default socket timeout is used.

Changed in version 3.3: ssl_context parameter was 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 *ssl_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: The optional *timeout* parameter was added.

The second subclass allows for connections created by a child process:

class imaplib.**IMAP4_stream**(*command*)

This is a subclass derived from IMAP4 that connects to the stdin/stdout file descriptors created by passing *command* to subprocess.Popen().

The following utility functions are defined:

imaplib.**Internaldate2tuple**(*datestr*)

Parse an IMAP4 INTERNALDATE string and return corresponding local time. The return value is a time.struct_time tuple or None if the string has wrong format.

imaplib.**Int2AP**(*num*)

Converts an integer into a bytes representation using characters from the set [A .. P].

imaplib.**ParseFlags**(*flagstr*)

Converts an IMAP4 FLAGS response to a tuple of individual flags.

imaplib.**Time2Internaldate**(*date_time*)

Convert *date_time* to an IMAP4 `INTERNALDATE` representation. The return value is a string in the form: `"DD-Mmm-YYYY HH:MM:SS +HHMM"` (including double-quotes). The *date_time* argument can be a number (int or float) representing seconds since epoch (as returned by `time.time()`), a 9-tuple representing local time an instance of `time.struct_time` (as returned by `time.localtime()`), an aware instance of `datetime.datetime`, or a double-quoted string. In the last case, it is assumed to already be in the correct format.

Note that IMAP4 message numbers change as the mailbox changes; in particular, after an `EXPUNGE` command performs deletions the remaining messages are renumbered. So it is highly advisable to use UIDs instead, with the UID command.

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

See also: Documents describing the protocol, sources for servers implementing it, by the University of Washington's IMAP Information Center can all be found at (**Source Code**) https://github.com/uw-imap/imap (**Not Maintained**).

IMAP4 Objects

All IMAP4rev1 commands are represented by methods of the same name, either upper-case or lower-case.

All arguments to commands are converted to strings, except for `AUTHENTICATE`, and the last argument to `APPEND` which is passed as an IMAP4 literal. If necessary (the string contains IMAP4 protocol-sensitive characters and isn't enclosed with either parentheses or double quotes) each string is quoted. However, the *password* argument to the `LOGIN` command is always quoted. If you want to avoid having an argument string quoted (eg: the *flags* argument to `STORE`) then enclose the string in parentheses (eg: `r'(\Deleted)'`).

Each command returns a tuple: `(type, [data, ...])` where *type* is usually `'OK'` or `'NO'`, and *data* is either the text from the command response, or mandated results from the command. Each *data* is either a `bytes`, or a tuple. If a tuple, then the first part is the header of the response, and the second part contains the data (ie: 'literal' value).

The *message_set* options to commands below is a string specifying one or more messages to be acted upon. It may be a simple message number (`'1'`), a range of message numbers (`'2:4'`), or a group of non-contiguous ranges separated by commas (`'1:3,6:9'`). A range can contain an asterisk to indicate an infinite upper bound (`'3:*'`).

An `IMAP4` instance has the following methods:

`IMAP4.`**`append`**(*`mailbox`*, *`flags`*, *`date_time`*, *`message`*)

Append *message* to named mailbox.

`IMAP4.`**`authenticate`**(*`mechanism`*, *`authobject`*)

Authenticate command — requires response processing.

mechanism specifies which authentication mechanism is to be used - it should appear in the instance variable `capabilities` in the form `AUTH=mechanism`.

authobject must be a callable object:

```
data = authobject(response)
```

It will be called to process server continuation responses; the *response* argument it is passed will be `bytes`. It should return `bytes` *data* that will be base64 encoded and sent to the server. It should return `None` if the client abort response `*` should be sent instead.

Changed in version 3.5: string usernames and passwords are now encoded to `utf-8` instead of being limited to ASCII.

`IMAP4.`**`check`**()

Checkpoint mailbox on server.

`IMAP4.`**`close`**()

Close currently selected mailbox. Deleted messages are removed from writable mailbox. This is the recommended command before `LOGOUT`.

`IMAP4.`**`copy`**(*`message_set`*, *`new_mailbox`*)

Copy *message_set* messages onto end of *new_mailbox*.

`IMAP4.`**`create`**(*`mailbox`*)

Create new mailbox named *mailbox*.

`IMAP4.`**`delete`**(*`mailbox`*)

Delete old mailbox named *mailbox*.

`IMAP4.`**`deleteacl`**(*`mailbox`*, *`who`*)

Delete the ACLs (remove any rights) set for who on mailbox.

`IMAP4.`**`enable`**(*`capability`*)

Enable *capability* (see **RFC 5161**). Most capabilities do not need to be enabled. Currently only the `UTF8=ACCEPT` capability is supported (see **RFC 6855**).

New in version 3.5: The `enable()` method itself, and **RFC 6855** support.

`IMAP4.`**`expunge`**()

Permanently remove deleted items from selected mailbox. Generates an `EXPUNGE` response for each deleted message. Returned data contains a list of `EXPUNGE` message numbers in order received.

`IMAP4.`**`fetch`**(*`message_set`*, *`message_parts`*)

Fetch (parts of) messages. *message_parts* should be a string of message part names enclosed within parentheses, eg: `"(UID BODY[TEXT])"`. Returned data are tuples of

message part envelope and data.

IMAP4.**getacl**(*mailbox*)

Get the ACLs for *mailbox*. The method is non-standard, but is supported by the Cyrus server.

IMAP4.**getannotation**(*mailbox*, *entry*, *attribute*)

Retrieve the specified ANNOTATIONs for *mailbox*. The method is non-standard, but is supported by the Cyrus server.

IMAP4.**getquota**(*root*)

Get the quota *root*'s resource usage and limits. This method is part of the IMAP4 QUOTA extension defined in rfc2087.

IMAP4.**getquotaroot**(*mailbox*)

Get the list of quota roots for the named *mailbox*. This method is part of the IMAP4 QUOTA extension defined in rfc2087.

IMAP4.**list**([*directory*[, *pattern*]])

List mailbox names in *directory* matching *pattern*. *directory* defaults to the top-level mail folder, and *pattern* defaults to match anything. Returned data contains a list of LIST responses.

IMAP4.**login**(*user*, *password*)

Identify the client using a plaintext password. The *password* will be quoted.

IMAP4.**login_cram_md5**(*user*, *password*)

Force use of CRAM-MD5 authentication when identifying the client to protect the password. Will only work if the server CAPABILITY response includes the phrase AUTH=CRAM-MD5.

IMAP4.**logout**()

Shutdown connection to server. Returns server BYE response.

Changed in version 3.8: The method no longer ignores silently arbitrary exceptions.

IMAP4.**lsub**(*directory='""'*, *pattern='*'*)

List subscribed mailbox names in directory matching pattern. *directory* defaults to the top level directory and *pattern* defaults to match any mailbox. Returned data are tuples of message part envelope and data.

IMAP4.**myrights**(*mailbox*)

Show my ACLs for a mailbox (i.e. the rights that I have on mailbox).

IMAP4.**namespace**()

Returns IMAP namespaces as defined in **RFC 2342**.

IMAP4.**noop**()

Send NOOP to server.

IMAP4.**open**(*host*, *port*, *timeout=None*)

Opens socket to *port* at *host*. The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If timeout is not given or is None, the global default socket timeout is used. Also note that if the *timeout* parameter is set to be zero, it will raise a ValueError to reject creating a non-blocking socket. This method is implicitly called by the IMAP4 constructor. The connection objects established by this method will be used in the IMAP4.read(), IMAP4.readline(), IMAP4.send(), and IMAP4.shutdown() methods. You may override this method.

Raises an auditing event imaplib.open with arguments self, host, port.

Changed in version 3.9: The *timeout* parameter was added.

IMAP4.**partial**(*message_num*, *message_part*, *start*, *length*)

Fetch truncated part of a message. Returned data is a tuple of message part envelope and data.

IMAP4.**proxyauth**(*user*)

Assume authentication as *user*. Allows an authorised administrator to proxy into any user's mailbox.

IMAP4.**read**(*size*)

Reads *size* bytes from the remote server. You may override this method.

IMAP4.**readline**()

Reads one line from the remote server. You may override this method.

IMAP4.**recent**()

Prompt server for an update. Returned data is None if no new messages, else value of RECENT response.

IMAP4.**rename**(*oldmailbox*, *newmailbox*)

Rename mailbox named *oldmailbox* to *newmailbox*.

IMAP4.**response**(*code*)

Return data for response *code* if received, or None. Returns the given code, instead of the usual type.

IMAP4.**search**(*charset*, *criterion*[, *...*])

Search mailbox for matching messages. *charset* may be None, in which case no CHARSET will be specified in the request to the server. The IMAP protocol requires that at least one criterion be specified; an exception will be raised when the server returns an error. *charset* must be None if the UTF8=ACCEPT capability was enabled using the enable() command.

Example:

```
# M is a connected IMAP4 instance...
typ, msgnums = M.search(None, 'FROM', '"LDJ"')

# or:
typ, msgnums = M.search(None, '(FROM "LDJ")')
```

IMAP4.**select**(*mailbox='INBOX'*, *readonly=False*)

Select a mailbox. Returned data is the count of messages in *mailbox* (EXISTS response). The default *mailbox* is 'INBOX'. If the *readonly* flag is set, modifications to the mailbox are not allowed.

IMAP4.**send**(*data*)

Sends data to the remote server. You may override this method.

Raises an auditing event imaplib.send with arguments self, data.

IMAP4.**setacl**(*mailbox*, *who*, *what*)

Set an ACL for *mailbox*. The method is non-standard, but is supported by the Cyrus server.

IMAP4.**setannotation**(*mailbox*, *entry*, *attribute*[, ...])

Set ANNOTATIONs for *mailbox*. The method is non-standard, but is supported by the Cyrus server.

IMAP4.**setquota**(*root*, *limits*)

Set the quota *root*'s resource *limits*. This method is part of the IMAP4 QUOTA extension defined in rfc2087.

IMAP4.**shutdown**()

Close connection established in open. This method is implicitly called by IMAP4.logout(). You may override this method.

IMAP4.**socket**()

Returns socket instance used to connect to server.

IMAP4.**sort**(*sort_criteria*, *charset*, *search_criterion*[, ...])

The sort command is a variant of search with sorting semantics for the results. Returned data contains a space separated list of matching message numbers.

Sort has two arguments before the *search_criterion* argument(s); a parenthesized list of *sort_criteria*, and the searching *charset*. Note that unlike search, the searching *charset* argument is mandatory. There is also a uid sort command which corresponds to sort the way that uid search corresponds to search. The sort command first searches the mailbox for messages that match the given searching criteria using the charset argument for the interpretation of strings in the searching criteria. It then returns the numbers of matching messages.

This is an IMAP4rev1 extension command.

IMAP4.**starttls**(*ssl_context=None*)

Send a STARTTLS command. The *ssl_context* argument is optional and should be a ssl.SSLContext object. This will enable encryption on the IMAP connection. Please read Security considerations for best practices.

New in version 3.2.

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

IMAP4.**status**(*mailbox*, *names*)

Request named status conditions for *mailbox*.

IMAP4.**store**(*message_set*, *command*, *flag_list*)

Alters flag dispositions for messages in mailbox. *command* is specified by section 6.4.6 of **RFC 2060** as being one of "FLAGS", "+FLAGS", or "-FLAGS", optionally with a suffix of ".SILENT".

For example, to set the delete flag on all messages:

```
typ, data = M.search(None, 'ALL')
for num in data[0].split():
   M.store(num, '+FLAGS', '\\Deleted')
M.expunge()
```

Note: Creating flags containing ']' (for example: "[test]") violates **RFC 3501** (the IMAP protocol). However, imaplib has historically allowed creation of such tags, and popular IMAP servers, such as Gmail, accept and produce such flags. There are non-Python programs which also create such tags. Although it is an RFC violation and IMAP clients and servers are supposed to be strict, imaplib nonetheless continues to allow such tags to be created for backward compatibility reasons, and as of Python 3.6, handles them if they are sent from the server, since this improves real-world compatibility.

IMAP4.**subscribe**(*mailbox*)

Subscribe to new mailbox.

IMAP4.**thread**(*threading_algorithm*, *charset*, *search_criterion*[, *...*])

The thread command is a variant of search with threading semantics for the results. Returned data contains a space separated list of thread members.

Thread members consist of zero or more messages numbers, delimited by spaces, indicating successive parent and child.

Thread has two arguments before the *search_criterion* argument(s); a *threading_algorithm*, and the searching *charset*. Note that unlike search, the searching *charset* argument is mandatory. There is also a uid thread command which corresponds to thread the way that uid search corresponds to search. The thread command first searches the mailbox for messages that match the given searching criteria

using the charset argument for the interpretation of strings in the searching criteria. It then returns the matching messages threaded according to the specified threading algorithm.

This is an `IMAP4rev1` extension command.

`IMAP4.`**`uid`**(*`command`*, *`arg`*[, `...`])

Execute command args with messages identified by UID, rather than message number. Returns response appropriate to command. At least one argument must be supplied; if none are provided, the server will return an error and an exception will be raised.

`IMAP4.`**`unsubscribe`**(*`mailbox`*)

Unsubscribe from old mailbox.

`IMAP4.`**`unselect`**()

`imaplib.IMAP4.unselect()` frees server's resources associated with the selected mailbox and returns the server to the authenticated state. This command performs the same actions as `imaplib.IMAP4.close()`, except that no messages are permanently removed from the currently selected mailbox.

New in version 3.9.

`IMAP4.`**`xatom`**(*`name`*[, `...`])

Allow simple extension commands notified by server in `CAPABILITY` response.

The following attributes are defined on instances of `IMAP4`:

`IMAP4.`**`PROTOCOL_VERSION`**

The most recent supported protocol in the `CAPABILITY` response from the server.

`IMAP4.`**`debug`**

Integer value to control debugging output. The initialize value is taken from the module variable `Debug`. Values greater than three trace each command.

`IMAP4.`**`utf8_enabled`**

Boolean value that is normally `False`, but is set to `True` if an `enable()` command is successfully issued for the `UTF8=ACCEPT` capability.

New in version 3.5.

IMAP4 Example

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

```
import getpass, imaplib

M = imaplib.IMAP4()
M.login(getpass.getuser(), getpass.getpass())
M.select()
```

```
typ, data = M.search(None, 'ALL')
for num in data[0].split():
    typ, data = M.fetch(num, '(RFC822)')
    print('Message %s\n%s\n' % (num, data[0][1]))
M.close()
M.logout()
```