imaplib — IMAP4 Client Library

Purpose: Client library for IMAP4 communication.

imaplib implements a client for communicating with Internet Message Access Protocol (IMAP) version 4 servers. The IMAP protocol defines a set of commands sent to the server and the responses delivered back to the client. Most of the commands are available as methods of the IMAP4 object used to communicate with the server.

These examples discuss part of the IMAP protocol, but are by no means complete. Refer to RFC 3501 for complete details.

Variations

There are three client classes for communicating with servers using various mechanisms. The first, IMAP4, uses clear text sockets; IMAP4_SSL uses encrypted communication over SSL sockets; and IMAP4_stream uses the standard input and standard output of an external command. All of the examples here will use IMAP4_SSL, but the APIs for the other classes are similar.

Connecting to a Server

There are two steps for establishing a connection with an IMAP server. First, set up the socket connection itself. Second, authenticate as a user with an account on the server. The following example code will read server and user information from a configuration file.

```
# imaplib_connect.py

import imaplib
import configparser
import os


def open_connection(verbose=False):
    # Read the config file
    config = configparser.ConfigParser()
    config.read([os.path.expanduser('~/.pymotw')])

    # Connect to the server
    hostname = config.get('server', 'hostname')
    if verbose:
        print('Connecting to', hostname)
    connection = imaplib.IMAP4_SSL(hostname)

    # Login to our account
    username = config.get('account', 'username')
    password = config.get('account', 'password')
    if verbose:
        print('Logging in as', username)
    connection.login(username, password)
    return connection


if __name__ == '__main__':
    with open_connection(verbose=True) as c:
        print(c)
```

When run, open_connection() reads the configuration information from a file in the user’s home directory, then opens the IMAP4_SSL connection and authenticates.

```
$ python3 imaplib_connect.py

Connecting to pymotw.hellfly.net
Logging in as example
<imaplib.IMAP4_SSL object at 0x10421e320>
```

The other examples in this section reuse this module, to avoid duplicating the code.

Authentication Failure

If the connection is established but authentication fails, an exception is raised.

```python
# imaplib_connect_fail.py

import imaplib
import configparser
import os

# Read the config file
config = configparser.ConfigParser()
config.read([os.path.expanduser('~/.pymotw')])

# Connect to the server
hostname = config.get('server', 'hostname')
print('Connecting to', hostname)
connection = imaplib.IMAP4_SSL(hostname)

# Login to our account
username = config.get('account', 'username')
password = 'this_is_the_wrong_password'
print('Logging in as', username)
try:
    connection.login(username, password)
except Exception as err:
    print('ERROR:', err)
```

This example uses the wrong password on purpose to trigger the exception.

```
$ python3 imaplib_connect_fail.py

Connecting to pymotw.hellfly.net
Logging in as example
ERROR: b'[AUTHENTICATIONFAILED] Authentication failed.'
```

Example Configuration

The example account has several mailboxes in a hierarchy:

- INBOX
- Deleted Messages
- Archive
- Example
    - 2016

There is one unread message in the `INBOX` folder, and one read message in `Example/2016`.

Listing Mailboxes

To retrieve the mailboxes available for an account, use the `list()` method.

```python
# imaplib_list.py

import imaplib
from pprint import pprint
from imaplib_connect import open_connection

with open_connection() as c:
    typ, data = c.list()
    print('Response code:', typ)
    print('Response:')
    pprint(data)
```

The return value is a `tuple` containing a response code and the data returned by the server. The response code is `OK`, unless there has been an error. The data for `list()` is a sequence of strings containing *flags*, the *hierarchy delimiter*, and *mailbox name* for each mailbox.

```
$ python3 imaplib_list.py

Response code: OK
Response:
[b'(\\HasChildren) "." Example',
 b'(\\HasNoChildren) "." Example.2016',
 b'(\\HasNoChildren) "." Archive',
 b'(\\HasNoChildren) "." "Deleted Messages"',
 b'(\\HasNoChildren) "." INBOX']
```

Each response string can be split into three parts using re or csv (see *IMAP Backup Script* in the references at the end of this section for an example using csv).

```
# imaplib_list_parse.py

import imaplib
import re

from imaplib_connect import open_connection

list_response_pattern = re.compile(
    r'\((?P<flags>.*?)\) "(?P<delimiter>.*)" (?P<name>.*)'
)


def parse_list_response(line):
    match = list_response_pattern.match(line.decode('utf-8'))
    flags, delimiter, mailbox_name = match.groups()
    mailbox_name = mailbox_name.strip('"')
    return (flags, delimiter, mailbox_name)


with open_connection() as c:
    typ, data = c.list()
print('Response code:', typ)

for line in data:
    print('Server response:', line)
    flags, delimiter, mailbox_name = parse_list_response(line)
    print('Parsed response:', (flags, delimiter, mailbox_name))
```

The server quotes the mailbox name if it includes spaces, but those quotes need to be stripped out to use the mailbox name in other calls back to the server later.

```
$ python3 imaplib_list_parse.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Parsed response: ('\\HasChildren', '.', 'Example')
Server response: b'(\\HasNoChildren) "." Example.2016'
Parsed response: ('\\HasNoChildren', '.', 'Example.2016')
Server response: b'(\\HasNoChildren) "." Archive'
Parsed response: ('\\HasNoChildren', '.', 'Archive')
Server response: b'(\\HasNoChildren) "." "Deleted Messages"'
Parsed response: ('\\HasNoChildren', '.', 'Deleted Messages')
Server response: b'(\\HasNoChildren) "." INBOX'
Parsed response: ('\\HasNoChildren', '.', 'INBOX')
```

`list()` takes arguments to specify mailboxes in part of the hierarchy. For example, to list sub-folders of Example, pass "Example" as the `directory` argument.

```
# imaplib_list_subfolders.py

import imaplib

from imaplib_connect import open_connection
```

```
with open_connection() as c:
    typ, data = c.list(directory='Example')

print('Response code:', typ)

for line in data:
    print('Server response:', line)
```

The parent and subfolder are returned.

```
$ python3 imaplib_list_subfolders.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Server response: b'(\\HasNoChildren) "." Example.2016'
```

Alternately, to list folders matching a pattern pass the `pattern` argument.

```
# imaplib_list_pattern.py

import imaplib

from imaplib_connect import open_connection

with open_connection() as c:
    typ, data = c.list(pattern='*Example*')

print('Response code:', typ)

for line in data:
    print('Server response:', line)
```

In this case, both `Example` and `Example.2016` are included in the response.

```
$ python3 imaplib_list_pattern.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Server response: b'(\\HasNoChildren) "." Example.2016'
```

Mailbox Status

Use `status()` to ask for aggregated information about the contents. the table below lists the status conditions defined by the standard.

IMAP 4 Mailbox Status Conditions

Condition	Meaning
MESSAGES	The number of messages in the mailbox.
RECENT	The number of messages with the \Recent flag set.
UIDNEXT	The next unique identifier value of the mailbox.
UIDVALIDITY	The unique identifier validity value of the mailbox.
UNSEEN	The number of messages which do not have the \Seen flag set.

The status conditions must be formatted as a space separated string enclosed in parentheses, the encoding for a "list" in the IMAP4 specification. The mailbox name is wrapped in " in case any of the names include spaces or other characters that would throw of the parser.

```
# imaplib_status.py

import imaplib
import re

from imaplib_connect import open_connection
from imaplib_list_parse import parse_list_response
from imaplib_list_parse import parse_list_response

with open_connection() as c:
    typ, data = c.list()
    for line in data:
        flags, delimiter, mailbox = parse_list_response(line)
        print('Mailbox:', mailbox)
        status = c.status(
            '"{}"'.format(mailbox),
            '(MESSAGES RECENT UIDNEXT UIDVALIDITY UNSEEN)',
        )
        print(status)
```

The return value is the usual tuple containing a response code and a list of information from the server. In this case, the list contains a single string formatted with the name of the mailbox in quotes, then the status conditions and values in parentheses.

```
$ python3 imaplib_status.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Parsed response: ('\\HasChildren', '.', 'Example')
Server response: b'(\\HasNoChildren) "." Example.2016'
Parsed response: ('\\HasNoChildren', '.', 'Example.2016')
Server response: b'(\\HasNoChildren) "." Archive'
Parsed response: ('\\HasNoChildren', '.', 'Archive')
Server response: b'(\\HasNoChildren) "." "Deleted Messages"'
Parsed response: ('\\HasNoChildren', '.', 'Deleted Messages')
Server response: b'(\\HasNoChildren) "." INBOX'
Parsed response: ('\\HasNoChildren', '.', 'INBOX')
Mailbox: Example
('OK', [b'Example (MESSAGES 0 RECENT 0 UIDNEXT 2 UIDVALIDITY 145
7297771 UNSEEN 0)'])
Mailbox: Example.2016
('OK', [b'Example.2016 (MESSAGES 1 RECENT 0 UIDNEXT 3 UIDVALIDIT
Y 1457297772 UNSEEN 0)'])
Mailbox: Archive
('OK', [b'Archive (MESSAGES 0 RECENT 0 UIDNEXT 1 UIDVALIDITY 145
7297770 UNSEEN 0)'])
Mailbox: Deleted Messages
('OK', [b'"Deleted Messages" (MESSAGES 3 RECENT 0 UIDNEXT 4 UIDV
ALIDITY 1457297773 UNSEEN 0)'])
Mailbox: INBOX
('OK', [b'INBOX (MESSAGES 2 RECENT 0 UIDNEXT 6 UIDVALIDITY 14572
97769 UNSEEN 1)'])
```

Selecting a Mailbox

The basic mode of operation, once the client is authenticated, is to select a mailbox, then interrogate the server regarding messages in the mailbox. The connection is stateful, so after a mailbox is selected all commands operate on messages in that mailbox until a new mailbox is selected.

```
# imaplib_select.py

import imaplib
import imaplib_connect

with imaplib_connect.open_connection() as c:
    typ, data = c.select('INBOX')
    print(typ, data)
    num_msgs = int(data[0])
    print('There are {} messages in INBOX'.format(num_msgs))
```

The response data contains the total number of messages in the mailbox.

```
$ python3 imaplib_select.py

OK [b'1']
There are 1 messages in INBOX
```

If an invalid mailbox is specified, the response code is NO.

```
# imaplib_select_invalid.py

import imaplib
import imaplib_connect

with imaplib_connect.open_connection() as c:
    typ, data = c.select('Does-Not-Exist')
    print(typ, data)
```

The data contains an error message describing the problem.

```
$ python3 imaplib_select_invalid.py

NO [b"Mailbox doesn't exist: Does-Not-Exist"]
```

Searching for Messages

After selecting the mailbox, use search() to retrieve the IDs of messages in the mailbox.

```
# imaplib_search_all.py

import imaplib
import imaplib_connect
from imaplib_list_parse import parse_list_response

with imaplib_connect.open_connection() as c:
    typ, mbox_data = c.list()
    for line in mbox_data:
        flags, delimiter, mbox_name = parse_list_response(line)
        c.select('"{}"'.format(mbox_name), readonly=True)
        typ, msg_ids = c.search(None, 'ALL')
        print(mbox_name, typ, msg_ids)
```

Message IDs are assigned by the server, and are implementation dependent. The IMAP4 protocol makes a distinction between sequential IDs for messages at a given point in time during a transaction and UID identifiers for messages, but not all servers implement both.

```
$ python3 imaplib_search_all.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Parsed response: ('\\HasChildren', '.', 'Example')
Server response: b'(\\HasNoChildren) "." Example.2016'
Parsed response: ('\\HasNoChildren', '.', 'Example.2016')
Server response: b'(\\HasNoChildren) "." Archive'
Parsed response: ('\\HasNoChildren', '.', 'Archive')
Server response: b'(\\HasNoChildren) "." "Deleted Messages"'
Parsed response: ('\\HasNoChildren', '.', 'Deleted Messages')
Server response: b'(\\HasNoChildren) "." INBOX'
Parsed response: ('\\HasNoChildren', '.', 'INBOX')
Example OK [b'']
Example.2016 OK [b'1']
Archive OK [b'']
Deleted Messages OK [b'']
INBOX OK [b'1']
```

In this case, INBOX and Example.2016 each have a different message with id 1. The other mailboxes are empty.

Search Criteria

A variety of other search criteria can be used, including looking at dates for the message, flags, and other headers. Refer to section 6.4.4. of RFC 3501 for complete details.

To look for messages with 'Example message 2' in the subject, the search criteria should be constructed as:

```
(SUBJECT "Example message 2")
```

This example finds all messages with the title “Example message 2” in all mailboxes:

```
# imaplib_search_subject.py

import imaplib
import imaplib_connect
from imaplib_list_parse import parse_list_response

with imaplib_connect.open_connection() as c:
    typ, mbox_data = c.list()
    for line in mbox_data:
        flags, delimiter, mbox_name = parse_list_response(line)
        c.select('"{}"'.format(mbox_name), readonly=True)
        typ, msg_ids = c.search(
            None,
            '(SUBJECT "Example message 2")',
        )
        print(mbox_name, typ, msg_ids)
```

There is only one such message in the account, and it is in the INBOX.

```
$ python3 imaplib_search_subject.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Parsed response: ('\\HasChildren', '.', 'Example')
Server response: b'(\\HasNoChildren) "." Example.2016'
Parsed response: ('\\HasNoChildren', '.', 'Example.2016')
Server response: b'(\\HasNoChildren) "." Archive'
Parsed response: ('\\HasNoChildren', '.', 'Archive')
Server response: b'(\\HasNoChildren) "." "Deleted Messages"'
Parsed response: ('\\HasNoChildren', '.', 'Deleted Messages')
Server response: b'(\\HasNoChildren) "." INBOX'
Parsed response: ('\\HasNoChildren', '.', 'INBOX')
Example OK [b'']
Example.2016 OK [b'']
Archive OK [b'']
Deleted Messages OK [b'']
INBOX OK [b'1']
```

Search criteria can also be combined.

```
# imaplib_search_from.py

import imaplib
import imaplib_connect
from imaplib_list_parse import parse_list_response

with imaplib_connect.open_connection() as c:
    typ, mbox_data = c.list()
    for line in mbox_data:
        flags, delimiter, mbox_name = parse_list_response(line)
        c.select('"{}"'.format(mbox_name), readonly=True)
        typ, msg_ids = c.search(
            None,
            '(FROM "Doug" SUBJECT "Example message 2")',
        )
        print(mbox_name, typ, msg_ids)
```

The criteria are combined with a logical and operation.

```
$ python3 imaplib_search_from.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Parsed response: ('\\HasChildren', '.', 'Example')
Server response: b'(\\HasNoChildren) "." Example.2016'
Server response: b'(\\HasNoChildren) "." Example.2016'
Parsed response: ('\\HasNoChildren', '.', 'Example.2016')
Server response: b'(\\HasNoChildren) "." Archive'
Parsed response: ('\\HasNoChildren', '.', 'Archive')
Server response: b'(\\HasNoChildren) "." "Deleted Messages"'
Parsed response: ('\\HasNoChildren', '.', 'Deleted Messages')
Server response: b'(\\HasNoChildren) "." INBOX'
Parsed response: ('\\HasNoChildren', '.', 'INBOX')
Example OK [b'']
Example.2016 OK [b'']
Archive OK [b'']
Deleted Messages OK [b'']
INBOX OK [b'1']
```

Fetching Messages

The identifiers returned by `search()` are used to retrieve the contents, or partial contents, of messages for further processing using the `fetch()` method. It takes two arguments, the message IDs to fetch and the portion(s) of the message to retrieve.

The `message_ids` argument is a comma separated list of ids (e.g., "1", "1,2") or ID ranges (e.g., 1:2). The `message_parts` argument is an IMAP list of message segment names. As with search criteria for `search()`, the IMAP protocol specifies named message segments so clients can efficiently retrieve only the parts of the message they actually need. For example, to retrieve the headers of the messages in a mailbox, use `fetch()` with the argument `BODY.PEEK[HEADER]`.

Note

Another way to fetch the headers is `BODY[HEADERS]`, but that form has a side-effect of implicitly marking the message as read, which is undesirable in many cases.

```python
# imaplib_fetch_raw.py

import imaplib
import pprint
import imaplib_connect

imaplib.Debug = 4
with imaplib_connect.open_connection() as c:
    c.select('INBOX', readonly=True)
    typ, msg_data = c.fetch('1', '(BODY.PEEK[HEADER] FLAGS)')
    pprint.pprint(msg_data)
```

The return value of `fetch()` has been partially parsed so it is somewhat harder to work with than the return value of `list()`. Turning on debugging shows the complete interaction between the client and server to understand why this is so.

```
$ python3 imaplib_fetch_raw.py

  19:40.68 imaplib version 2.58
  19:40.68 new IMAP4 connection, tag=b'IIEN'
  19:40.70 < b'* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN
-REFERRALS ID ENABLE IDLE AUTH=PLAIN] Dovecot (Ubuntu) ready.'
  19:40.70 > b'IIEN0 CAPABILITY'
  19:40.73 < b'* CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REF
ERRALS ID ENABLE IDLE AUTH=PLAIN'
  19:40.73 < b'IIEN0 OK Pre-login capabilities listed, post-logi
n capabilities have more.'
  19:40.73 CAPABILITIES: ('IMAP4REV1', 'LITERAL+', 'SASL-IR', 'L
OGIN-REFERRALS', 'ID', 'ENABLE', 'IDLE', 'AUTH=PLAIN')
  19:40.73 > b'IIEN1 LOGIN example "TMFw00fpymotw"'
  19:40.79 < b'* CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REF
ERRALS ID ENABLE IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD
=REFS THREAD=ORDEREDSUBJECT MULTIAPPEND URL-PARTIAL CATENATE UNS
ELECT CHILDREN NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CONDS
TORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARCH LIST-
STATUS SPECIAL-USE BINARY MOVE'
  19:40.79 < b'IIEN1 OK Logged in'
  19:40.79 > b'IIEN2 EXAMINE INBOX'
  19:40.82 < b'* FLAGS (\\Answered \\Flagged \\Deleted \\Seen \\
Draft)'
  19:40.82 < b'* OK [PERMANENTFLAGS ()] Read-only mailbox.'
  19:40.82 < b'* OK [PERMANENTFLAGS ()] Read-only mailbox.'
  19:40.82 < b'* 2 EXISTS'
  19:40.82 < b'* 0 RECENT'
  19:40.82 < b'* OK [UNSEEN 1] First unseen.'
  19:40.82 < b'* OK [UIDVALIDITY 1457297769] UIDs valid'
  19:40.82 < b'* OK [UIDNEXT 6] Predicted next UID'
  19:40.82 < b'* OK [HIGHESTMODSEQ 20] Highest'
  19:40.82 < b'IIEN2 OK [READ-ONLY] Examine completed (0.000 sec
s).'
  19:40.82 > b'IIEN3 FETCH 1 (BODY.PEEK[HEADER] FLAGS)'
  19:40.86 < b'* 1 FETCH (FLAGS () BODY[HEADER] {3108}'
  19:40.86 read literal size 3108
  19:40.86 < b')'
  19:40.89 < b'IIEN3 OK Fetch completed.'
  19:40.89 > b'IIEN4 LOGOUT'
  19:40.93 < b'* BYE Logging out'
  19:40.93 BYE response: b'Logging out'
[(b'1 (FLAGS () BODY[HEADER] {3108}',
  b'Return-Path: <doug@doughellmann.com>\r\nReceived: from compu
te4.internal ('
  b'compute4.nyi.internal [10.202.2.44])\r\n\t by sloti26t01 (Cy
rus 3.0.0-beta1'
  b'-git-fastmail-12410) with LMTPA;\r\n\t Sun, 06 Mar 2016 16:1
6:03 -0500\r'
  b'\nX-Sieve: CMU Sieve 2.4\r\nX-Spam-known-sender: yes, fadd1c
f2-dc3a-4984-a0'
  b'8b-02cef3cf1221="doug",\r\n  ea349ad0-9299-47b5-b632-6ff1e39
4cc7d="both he'
  b'llfly"\r\nX-Spam-score: 0.0\r\nX-Spam-hits: ALL_TRUSTED -1,
BAYES_00 -1.'
  b'9, LANGUAGES unknown, BAYES_USED global,\r\n  SA_VERSION 3.3
.2\r\nX-Spam'
  b"-source: IP='127.0.0.1', Host='unk', Country='unk', FromHead
er='com',\r\n "
  b" MailFrom='com'\r\nX-Spam-charsets: plain='us-ascii'\r\nX-Re
solved-to: d"
  b'oughellmann@fastmail.fm\r\nX-Delivered-to: doug@doughellmann
.com\r\nX-Ma'
  b'il-from: doug@doughellmann.com\r\nReceived: from mx5 ([10.20
2.2.204])\r'
  b'\n  by compute4.internal (LMTPProxy); Sun, 06 Mar 2016 16:16
:03 -0500\r\nRe'
  b'ceived: from mx5.nyi.internal (localhost [127.0.0.1])\r\n\tb
y mx5.nyi.inter'
  b'nal (Postfix) with ESMTP id 47CBA280DB3\r\n\tfor <doug@dough
ellmann.com>; S'
  b'un,  6 Mar 2016 16:16:03 -0500 (EST)\r\nReceived: from mx5.n
yi.internal (l'
  b'ocalhost [127.0.0.1])\r\n    by mx5.nyi.internal (Authentica
tion Milter) w'
  b'ith ESMTP\r\n    id A717886846E.30BA4280D81;\r\n    Sun, 6 M
ar 2016 16:1'
  b'6:03 -0500\r\nAuthentication-Results: mx5.nyi.internal;\r\n
   dkim=pass'
  b' (1024-bit rsa key) header.d=messagingengine.com header.i=@m
essagingengi'
  b'ne.com header.b=Jrsm+pCo;\r\n    x-local-ip=pass\r\nReceived
: from mailo'
  b'ut.nyi.internal (gateway1.nyi.internal [10.202.2.221])\r\n\t
(using TLSv1.2 '
  b'with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/256 bits))\r\n\
t(No client cer'
  b'tificate requested)\r\n\tby mx5.nyi.internal (Postfix) with
ESMTPS id 30BA4'
  b'280D81\r\n\tfor <doug@doughellmann.com>; Sun,  6 Mar 2016 16
:16:03 -0500 (E'
  b'ST)\r\nReceived: from compute2.internal (compute2.nyi.intern
al [10.202.2.4'
  b'2])\r\n\tby mailout.nyi.internal (Postfix) with ESMTP id 174
0420D0A\r\n\tf'
  b'or <doug@doughellmann.com>; Sun,  6 Mar 2016 16:16:03 -0500
(EST)\r\nRecei'
```

```
  b'ved: from frontend2 ([10.202.2.161])\r\n  by compute2.intern
al (MEProxy); '
  b'Sun, 06 Mar 2016 16:16:03 -0500\r\nDKIM-Signature: v=1; a=rs
a-sha1; c=rela'
  b'xed/relaxed; d=\r\n\tmessagingengine.com; h=content-transfer
-encoding:conte'
  b'nt-type\r\n\t:date:from:message-id:mime-version:subject:to:x
-sasl-enc\r\n'
  b'\t:x-sasl-enc; s=smtpout; bh=P98NTsEo015suwJ4gk71knAWLa4=; b
=Jrsm+\r\n\t'
  b'pCovRIoQIRyp8Fl0L6JHOI8sbZy2obx7O28JF2iTlTWmX33Rhlq9403XRklw
N3JA\r\n\t7KSPq'
  b'MTp30Qdx6yIUaADwQqlO+QMuQq/QxBHdjeebmdhgVfjhqxrzTbSMww/ZNhL\
r\n\tYwv/QM/oDH'
  b'bXiLSUlB3Qrg+9wsE/0jU/EOisiU=\r\nX-Sasl-enc: 8ZJ+4ZRE8AGPzdL
RWQFivGymJb8pa'
  b'4G9JGcb7k4xKn+I 1457298962\r\nReceived: from [192.168.1.14]
(75-137-1-34.d'
  b'hcp.nwnn.ga.charter.com [75.137.1.34])\r\n\tby mail.messagin
gengine.com (Po'
  b'stfix) with ESMTPA id C0B366801CD\r\n\tfor <doug@doughellman
n.com>; Sun,  6'
  b' Mar 2016 16:16:02 -0500 (EST)\r\nFrom: Doug Hellmann <doug@
doughellmann.c'
  b'om>\r\nContent-Type: text/plain; charset=us-ascii\r\nContent
-Transfer-En'
  b'coding: 7bit\r\nSubject: PyMOTW Example message 2\r\nMessage
-Id: <00ABCD'
  b'46-DADA-4912-A451-D27165BC3A2F@doughellmann.com>\r\nDate: Su
n, 6 Mar 2016 '
  b'16:16:02 -0500\r\nTo: Doug Hellmann <doug@doughellmann.com>\
r\nMime-Vers'
  b'ion: 1.0 (Mac OS X Mail 9.2 \\(3112\\))\r\nX-Mailer: Apple M
ail (2.3112)'
  b'\r\n\r\n'),
 b')']
```

The response from the FETCH command starts with the flags, then indicates that there are 595 bytes of header data. The client constructs a tuple with the response for the message, and then closes the sequence with a single string containing the right parenthesis (")") the server sends at the end of the fetch response. Because of this formatting, it may be easier to fetch different pieces of information separately, or to recombine the response and parse it in the client.

```
# imaplib_fetch_separately.py

import imaplib
import pprint
import imaplib_connect

with imaplib_connect.open_connection() as c:
    c.select('INBOX', readonly=True)

    print('HEADER:')
    typ, msg_data = c.fetch('1', '(BODY.PEEK[HEADER])')
    for response_part in msg_data:
        if isinstance(response_part, tuple):
            print(response_part[1])

    print('\nBODY TEXT:')
    typ, msg_data = c.fetch('1', '(BODY.PEEK[TEXT])')
    for response_part in msg_data:
        if isinstance(response_part, tuple):
            print(response_part[1])

    print('\nFLAGS:')
    typ, msg_data = c.fetch('1', '(FLAGS)')
    for response_part in msg_data:
        print(response_part)
        print(imaplib.ParseFlags(response_part))
```

Fetching values separately has the added benefit of making it easy to use `ParseFlags()` to parse the flags from the response.

```
$ python3 imaplib_fetch_separately.py

HEADER:
b'Return-Path: <doug@doughellmann.com>\r\nReceived: from compute
4.internal (compute4.nyi.internal [10.202.2.44])\r\n\t by sloti2
6t01 (Cyrus 3.0.0-beta1-git-fastmail-12410) with LMTPA;\r\n\t Su
n, 06 Mar 2016 16:16:03 -0500\r\nX-Sieve: CMU Sieve 2.4\r\nX-Spa
m-known-sender: yes, fadd1cf2-dc3a-4984-a08b-02cef3cf1221="doug"
,\r\n  ea349ad0-9299-47b5-b632-6ff1e394cc7d="both hellfly"\r\nX-
Spam-score: 0.0\r\nX-Spam-hits: ALL_TRUSTED -1, BAYES_00 -1.9, L
ANGUAGES unknown, BAYES_USED global,\r\n  SA_VERSION 3.3.2\r\nX-
Spam-source: IP=\'127.0.0.1\', Host=\'unk\', Country=\'unk\', Fr
omHeader=\'com\',\r\n  MailFrom=\'com\'\r\nX-Spam-charsets: plai
n=\'us-ascii\'\r\nX-Resolved-to: doughellmann@fastmail.fm\r\nX-D
elivered-to: doug@doughellmann.com\r\nX-Mail-from: doug@doughell
mann.com\r\nReceived: from mx5 ([10.202.2.204])\r\n  by compute4
.internal (LMTPProxy); Sun, 06 Mar 2016 16:16:03 -0500\r\nReceiv
ed: from mx5.nyi.internal (localhost [127.0.0.1])\r\n\tby mx5.ny
i.internal (Postfix) with ESMTP id 47CBA280DB3\r\n\tfor <doug@do
ughellmann.com>; Sun,  6 Mar 2016 16:16:03 -0500 (EST)\r\nReceiv
ed: from mx5.nyi.internal (localhost [127.0.0.1])\r\n    by mx5.
nyi.internal (Authentication Milter) with ESMTP\r\n    id A71788
6846E.30BA4280D81;\r\n    Sun, 6 Mar 2016 16:16:03 -0500\r\nAuth
entication-Results: mx5.nyi.internal;\r\n    dkim=pass (1024-bit
 rsa key) header.d=messagingengine.com header.i=@messagingengine
.com header.b=Jrsm+pCo;\r\n    x-local-ip=pass\r\nReceived: from
 mailout.nyi.internal (gateway1.nyi.internal [10.202.2.221])\r\n
\t(using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/25
6 bits))\r\n\t(No client certificate requested)\r\n\tby mx5.nyi.
internal (Postfix) with ESMTPS id 30BA4280D81\r\n\tfor <doug@dou
ghellmann.com>; Sun,  6 Mar 2016 16:16:03 -0500 (EST)\r\nReceive
d: from compute2.internal (compute2.nyi.internal [10.202.2.42])\
r\n\tby mailout.nyi.internal (Postfix) with ESMTP id 1740420D0A\
r\n\tfor <doug@doughellmann.com>; Sun,  6 Mar 2016 16:16:03 -050
0 (EST)\r\nReceived: from frontend2 ([10.202.2.161])\r\n  by com
pute2.internal (MEProxy); Sun, 06 Mar 2016 16:16:03 -0500\r\nDKI
M-Signature: v=1; a=rsa-sha1; c=relaxed/relaxed; d=\r\n\tmessagi
ngengine.com; h=content-transfer-encoding:content-type\r\n\t:dat
e:from:message-id:mime-version:subject:to:x-sasl-enc\r\n\t:x-sas
l-enc; s=smtpout; bh=P98NTsEo015suwJ4gk71knAWLa4=; b=Jrsm+\r\n\t
pCovRIoQIRyp8Fl0L6JHOI8sbZy2obx7O28JF2iTlTWmX33Rhlq9403XRklwN3JA
\r\n\t7KSPqMTp30Qdx6yIUaADwQqlO+QMuQq/QxBHdjeebmdhgVfjhqxrzTbSMw
w/ZNhL\r\n\tYwv/QM/oDHbXiLSUlB3Qrg+9wsE/0jU/EOisiU=\r\nX-Sasl-en
c: 8ZJ+4ZRE8AGPzdLRWQFivGymJb8pa4G9JGcb7k4xKn+I 1457298962\r\nRe
ceived: from [192.168.1.14] (75-137-1-34.dhcp.nwnn.ga.charter.co
m [75.137.1.34])\r\n\tby mail.messagingengine.com (Postfix) with
 ESMTPA id C0B366801CD\r\n\tfor <doug@doughellmann.com>; Sun,  6
 Mar 2016 16:16:02 -0500 (EST)\r\nFrom: Doug Hellmann <doug@doug
hellmann.com>\r\nContent-Type: text/plain; charset=us-ascii\r\nC
ontent-Transfer-Encoding: 7bit\r\nSubject: PyMOTW Example messag
e 2\r\nMessage-Id: <00ABCD46-DADA-4912-A451-D27165BC3A2F@doughel
lmann.com>\r\nDate: Sun, 6 Mar 2016 16:16:02 -0500\r\nTo: Doug H
ellmann <doug@doughellmann.com>\r\nMime-Version: 1.0 (Mac OS X M
ail 9.2 \\(3112\\))\r\nX-Mailer: Apple Mail (2.3112)\r\n\r\n'

BODY TEXT:
b'This is the second example message.\r\n'

FLAGS:
b'1 (FLAGS ())'
()
```

Whole Messages

As illustrated earlier, the client can ask the server for individual parts of the message separately. It is also possible to retrieve the entire message as an RFC 822 formatted mail message and parse it with classes from the email module.

```
# imaplib_fetch_rfc822.py

import imaplib
import imaplib
import email
import email.parser

import imaplib_connect


with imaplib_connect.open_connection() as c:
    c.select('INBOX', readonly=True)

    typ, msg_data = c.fetch('1', '(RFC822)')
    for response_part in msg_data:
        if isinstance(response_part, tuple):
            email_parser = email.parser.BytesFeedParser()
            email_parser.feed(response_part[1])
            msg = email_parser.close()
            for header in ['subject', 'to', 'from']:
                print('{:^8}: {}'.format(
                    header.upper(), msg[header]))
```

The parser in the email module make it very easy to access and manipulate messages. This example prints just a few of the headers for each message.

```
$ python3 imaplib_fetch_rfc822.py

SUBJECT : PyMOTW Example message 2
   TO   : Doug Hellmann <doug@doughellmann.com>
  FROM  : Doug Hellmann <doug@doughellmann.com>
```

Uploading Messages

To add a new message to a mailbox, construct a Message instance and pass it to the append() method, along with the timestamp for the message.

```
# imaplib_append.py

import imaplib
import time
import email.message
import imaplib_connect

new_message = email.message.Message()
new_message.set_unixfrom('pymotw')
new_message['Subject'] = 'subject goes here'
new_message['From'] = 'pymotw@example.com'
new_message['To'] = 'example@example.com'
new_message.set_payload('This is the body of the message.\n')

print(new_message)

with imaplib_connect.open_connection() as c:
    c.append('INBOX', '',
             imaplib.Time2Internaldate(time.time()),
             str(new_message).encode('utf-8'))

    # Show the headers for all messages in the mailbox
    c.select('INBOX')
    typ, [msg_ids] = c.search(None, 'ALL')
    for num in msg_ids.split():
        typ, msg_data = c.fetch(num, '(BODY.PEEK[HEADER])')
        for response_part in msg_data:
            if isinstance(response_part, tuple):
                print('\n{}:'.format(num))
                print(response_part[1])
```

The payload used in this example is a simple plaintext email body. Message also supports MIME-encoded multi-part messages.

```
$ python3 imaplib_append.py

```

```
Subject: subject goes here
From: pymotw@example.com
To: example@example.com

This is the body of the message.


b'1':
b'Return-Path: <doug@doughellmann.com>\r\nReceived: from compute
4.internal (compute4.nyi.internal [10.202.2.44])\r\n\t by sloti2
6t01 (Cyrus 3.0.0-beta1-git-fastmail-12410) with LMTPA;\r\n\t Su
n, 06 Mar 2016 16:16:03 -0500\r\nX-Sieve: CMU Sieve 2.4\r\nX-Spa
m-known-sender: yes, fadd1cf2-dc3a-4984-a08b-02cef3cf1221="doug"
,\r\n  ea349ad0-9299-47b5-b632-6ff1e394cc7d="both hellfly"\r\nX-
Spam-score: 0.0\r\nX-Spam-hits: ALL_TRUSTED -1, BAYES_00 -1.9, L
ANGUAGES unknown, BAYES_USED global,\r\n  SA_VERSION 3.3.2\r\nX-
Spam-source: IP=\'127.0.0.1\', Host=\'unk\', Country=\'unk\', Fr
omHeader=\'com\',\r\n  MailFrom=\'com\'\r\nX-Spam-charsets: plai
n=\'us-ascii\'\r\nX-Resolved-to: doughellmann@fastmail.fm\r\nX-D
elivered-to: doug@doughellmann.com\r\nX-Mail-from: doug@doughell
mann.com\r\nReceived: from mx5 ([10.202.2.204])\r\n  by compute4
.internal (LMTPProxy); Sun, 06 Mar 2016 16:16:03 -0500\r\nReceiv
ed: from mx5.nyi.internal (localhost [127.0.0.1])\r\n\tby mx5.ny
i.internal (Postfix) with ESMTP id 47CBA280DB3\r\n\tfor <doug@do
ughellmann.com>; Sun,  6 Mar 2016 16:16:03 -0500 (EST)\r\nReceiv
ed: from mx5.nyi.internal (localhost [127.0.0.1])\r\n    by mx5.
nyi.internal (Authentication Milter) with ESMTP\r\n    id A71788
6846E.30BA4280D81;\r\n    Sun, 6 Mar 2016 16:16:03 -0500\r\nAuth
entication-Results: mx5.nyi.internal;\r\n    dkim=pass (1024-bit
 rsa key) header.d=messagingengine.com header.i=@messagingengine
.com header.b=Jrsm+pCo;\r\n    x-local-ip=pass\r\nReceived: from
 mailout.nyi.internal (gateway1.nyi.internal [10.202.2.221])\r\n
\t(using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/25
6 bits))\r\n\t(No client certificate requested)\r\n\tby mx5.nyi.
internal (Postfix) with ESMTPS id 30BA4280D81\r\n\tfor <doug@dou
ghellmann.com>; Sun,  6 Mar 2016 16:16:03 -0500 (EST)\r\nReceive
d: from compute2.internal (compute2.nyi.internal [10.202.2.42])\
r\n\tby mailout.nyi.internal (Postfix) with ESMTP id 1740420D0A\
r\n\tfor <doug@doughellmann.com>; Sun,  6 Mar 2016 16:16:03 -050
0 (EST)\r\nReceived: from frontend2 ([10.202.2.161])\r\n  by com
pute2.internal (MEProxy); Sun, 06 Mar 2016 16:16:03 -0500\r\nDKI
M-Signature: v=1; a=rsa-sha1; c=relaxed/relaxed; d=\r\n\tmessagi
ngengine.com; h=content-transfer-encoding:content-type\r\n\t:dat
e:from:message-id:mime-version:subject:to:x-sasl-enc\r\n\t:x-sas
l-enc; s=smtpout; bh=P98NTsEo015suwJ4gk71knAWLa4=; b=Jrsm+\r\n\t
pCovRIoQIRyp8Fl0L6JHOI8sbZy2obx7O28JF2iTlTWmX33Rhlq9403XRklwN3JA
\r\n\t7KSPqMTp30Qdx6yIUaADwQqlO+QMuQq/QxBHdjeebmdhgVfjhqxrzTbSMw
w/ZNhL\r\n\tYwv/QM/oDHbXiLSUlB3Qrg+9wsE/0jU/EOisiU=\r\nX-Sasl-en
c: 8ZJ+4ZRE8AGPzdLRWQFivGymJb8pa4G9JGcb7k4xKn+I 1457298962\r\nRe
ceived: from [192.168.1.14] (75-137-1-34.dhcp.nwnn.ga.charter.co
m [75.137.1.34])\r\n\tby mail.messagingengine.com (Postfix) with
 ESMTPA id C0B366801CD\r\n\tfor <doug@doughellmann.com>; Sun,  6
 Mar 2016 16:16:02 -0500 (EST)\r\nFrom: Doug Hellmann <doug@doug
hellmann.com>\r\nContent-Type: text/plain; charset=us-ascii\r\nC
ontent-Transfer-Encoding: 7bit\r\nSubject: PyMOTW Example messag
e 2\r\nMessage-Id: <00ABCD46-DADA-4912-A451-D27165BC3A2F@doughel
lmann.com>\r\nDate: Sun, 6 Mar 2016 16:16:02 -0500\r\nTo: Doug H
ellmann <doug@doughellmann.com>\r\nMime-Version: 1.0 (Mac OS X M
ail 9.2 \\(3112\\))\r\nX-Mailer: Apple Mail (2.3112)\r\n\r\n'

b'2':
b'Subject: subject goes here\r\nFrom: pymotw@example.com\r\nTo:
example@example.com\r\n\r\n'
```

Moving and Copying Messages

Once a message is on the server, it can be moved or copied without downloading it using `move()` or `copy()`. These methods operate on message id ranges, just as `fetch()` does.

```
# imaplib_archive_read.py
```

```
import imaplib
import imaplib_connect

with imaplib_connect.open_connection() as c:
    # Find the "SEEN" messages in INBOX
    c.select('INBOX')
    typ, [response] = c.search(None, 'SEEN')
    if typ != 'OK':
        raise RuntimeError(response)
    msg_ids = ','.join(response.decode('utf-8').split(' '))

    # Create a new mailbox, "Example.Today"
    typ, create_response = c.create('Example.Today')
    print('CREATED Example.Today:', create_response)

    # Copy the messages
    print('COPYING:', msg_ids)
    c.copy(msg_ids, 'Example.Today')

    # Look at the results
    c.select('Example.Today')
    typ, [response] = c.search(None, 'ALL')
    print('COPIED:', response)
```

This example script creates a new mailbox under `Example` and copies the read messages from `INBOX` into it.

```
$ python3 imaplib_archive_read.py

CREATED Example.Today: [b'Completed']
COPYING: 2
COPIED: b'1'
```

Running the same script again shows the importance to checking return codes. Instead of raising an exception, the call to `create()` to make the new mailbox reports that the mailbox already exists.

```
$ python3 imaplib_archive_read.py

CREATED Example.Today: [b'[ALREADYEXISTS] Mailbox already exists
']
COPYING: 2
COPIED: b'1 2'
```

Deleting Messages

Although many modern mail clients use a “Trash folder” model for working with deleted messages, the messages are not usually moved into an actual folder. Instead, their flags are updated to add `\Deleted`. The operation for “emptying” the trash is implemented through the `EXPUNGE` command. This example script finds the archived messages with “Lorem ipsum” in the subject, sets the deleted flag, then shows that the messages are still present in the folder by querying the server again.

```
# imaplib_delete_messages.py

import imaplib
import imaplib_connect
from imaplib_list_parse import parse_list_response

with imaplib_connect.open_connection() as c:
    c.select('Example.Today')

    # What ids are in the mailbox?
    typ, [msg_ids] = c.search(None, 'ALL')
    print('Starting messages:', msg_ids)

    # Find the message(s)
    typ, [msg_ids] = c.search(
        None,
        '(SUBJECT "subject goes here")',
    )
    msg_ids = ','.join(msg_ids.decode('utf-8').split(' '))
    print('Matching messages:', msg_ids)

    # What are the current flags?
    typ, response = c.fetch(msg_ids, '(FLAGS)')
    print('Flags before:', response)

    # Change the Deleted flag
    typ, response = c.store(msg_ids, '+FLAGS', r'(\Deleted)')

    # What are the flags now?
    typ, response = c.fetch(msg_ids, '(FLAGS)')
    print('Flags after:', response)

    # Really delete the message.
    typ, response = c.expunge()
    print('Expunged:', response)

    # What ids are left in the mailbox?
    typ, [msg_ids] = c.search(None, 'ALL')
    print('Remaining messages:', msg_ids)
```

Explicitly calling `expunge()` removes the messages, but calling `close()` has the same effect. The difference is the client is not notified about the deletions when `close()` is called.

```
$ python3 imaplib_delete_messages.py

Response code: OK
Server response: b'(\\HasChildren) "." Example'
Parsed response: ('\\HasChildren', '.', 'Example')
Server response: b'(\\HasNoChildren) "." Example.Today'
Parsed response: ('\\HasNoChildren', '.', 'Example.Today')
Server response: b'(\\HasNoChildren) "." Example.2016'
Parsed response: ('\\HasNoChildren', '.', 'Example.2016')
Server response: b'(\\HasNoChildren) "." Archive'
Parsed response: ('\\HasNoChildren', '.', 'Archive')
Server response: b'(\\HasNoChildren) "." "Deleted Messages"'
Parsed response: ('\\HasNoChildren', '.', 'Deleted Messages')
Server response: b'(\\HasNoChildren) "." INBOX'
Parsed response: ('\\HasNoChildren', '.', 'INBOX')
Starting messages: b'1 2'
Matching messages: 1,2
Flags before: [b'1 (FLAGS (\\Seen))', b'2 (FLAGS (\\Seen))']
Flags after: [b'1 (FLAGS (\\Deleted \\Seen))', b'2 (FLAGS (\\Del
eted \\Seen))']
Expunged: [b'2', b'1']
Remaining messages: b''
```

See also

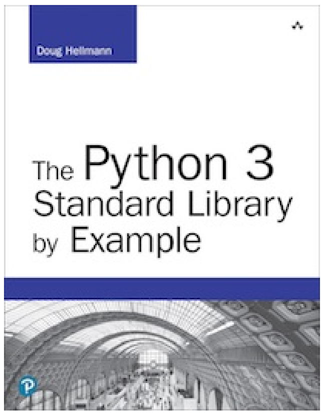
- Standard library documentation for imaplib
- `rfc822` – The `rfc822` module includes an RFC 822 / RFC 5322 parser.
- `email` – The `email` module for parsing email messages.
- mailbox – Local mailbox parser.
- `ConfigParser` – Read and write configuration files.
- University of Washington IMAP Information Center – Good resource for IMAP information, along with source code.
- **RFC 3501** – Internet Message Access Protocol
- **RFC 5322** – Internet Message Format
- IMAP Backup Script – A script to backup email from an IMAP server.
- IMAPClient – A higher-level client for talking to IMAP servers, written by Menno Smits.
- offlineimap – A Python application for keeping a local set of mailboxes in sync with an IMAP server.
- Python 2 to 3 porting notes for imaplib

mailbox — Manipulate Email Archives   Application Building Blocks

This page was last updated 2016-12-29.

Navigation

mailbox — Manipulate Email Archives
Application Building Blocks



The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index