IMAPClient Documentation

Release 2.2.0-alpha

Menno Smits

Contents

Author Menno Finlay-Smits

Version 2.2.0-alpha

Date Sep 19, 2018

Homepage http://imapclient.freshfoo.com

Download http://pypi.python.org/pypi/IMAPClient/

Source code https://github.com/mjs/imapclient

Documentation http://imapclient.readthedocs.io/

License New BSD License

Support Mailing List

Introduction

IMAPClient is an easy-to-use, Pythonic and complete IMAP client library.

Although IMAPClient actually uses the imaplib module from the Python standard library under the hood, it provides a different API. Instead of requiring that the caller performs extra parsing work, return values are full parsed, readily usable and use sensible Python types. Exceptions are raised when problems occur (no error checking of return values is required).

IMAPClient is straight forward it use, but it can be useful to have at least a general understanding of the IMAP protocol. RFC 3501 explains IMAP in detail. Other RFCs also apply to various extensions to the base protocol. These are referred to in the documentation below where relevant.

Python versions 2.7 and 3.4 through 3.7 are officially supported.

Getting Started

Install IMAPClient:

```
$ pip install imapclient
```

See *Installation* for more details.

The core of the IMAPClient API is the IMAPClient class. Instantiating this class, creates a connection to an IMAP account. Calling methods on the IMAPClient instance interacts with the server.

The following example shows a simple interaction with an IMAP server. It displays the message ID, subject and date of the message for all messages in the INBOX folder.

```
>>> from imapclient import IMAPClient
>>> server = IMAPClient('imap.mailserver.com', use_uid=True)
>>> server.login('someuser', 'somepassword')
b'[CAPABILITY IMAP4rev1 LITERAL+ SASL-IR [...] LIST-STATUS QUOTA] Logged in'

>>> select_info = server.select_folder('INBOX')
>>> print('%d messages in INBOX' % select_info[b'EXISTS'])
34 messages in INBOX

>>> messages = server.search(['FROM', 'best-friend@domain.com'])
>>> print("%d messages from our best friend" % len(messages))
5 messages from our best friend

>>> for msgid, data in server.fetch(messages, ['ENVELOPE']).items():
>>>     envelope = data[b'ENVELOPE']
>>>     print('ID #%d: "%s" received %s' % (msgid, envelope.subject.decode(),
→envelope.date))
ID #62: "Our holidays photos" received 2017-07-20 21:47:42
ID #55: "Re: did you book the hotel?" received 2017-06-26 10:38:09
ID #53: "Re: did you book the hotel?" received 2017-06-25 22:02:58
ID #44: "See that fun article about lobsters in Pacific ocean!" received 2017-06-09
→09:49:47
ID #46: "Planning for our next vacations" received 2017-05-12 10:29:30
```

(continues on next page)

```
>>> server.logout()
b'Logging out'
```

User Guide

This section describes how IMAPClient works and gives some examples to help you start.

3.1 Installation

3.1.1 Pip

IMAPClient can easily be installed with pip:

```
$ pip install imapclient
```

3.1.2 From Source

IMAPClient is developed on GitHub, you can find the code at mjs/imapclient.

You can clone the public repository:

```
$ git clone https://github.com/mjs/imapclient.git
```

Once you have the sources, simply install IMAPClient with:

```
$ cd imapclient
$ pip install -e .
```

3.1.3 Other versions

The source distributions of all IMAPClient versions are available at http://menno.io/projects/IMAPClient/. Alternatively you can also use the PyPI page at https://pypi.python.org/pypi/IMAPClient/.

3.2 IMAPClient Concepts

3.2.1 Message Identifiers

In the IMAP protocol, messages are identified using an integer. These message ids are specific to a given folder.

There are two types of message identifiers in the IMAP protocol.

One type is the message sequence number where the messages in a folder are numbered from 1 to N where N is the number of messages in the folder. These numbers don't persist between sessions and may be reassigned after some operations such as an expunge.

A more convenient approach is Unique Identifiers (UIDs). Unique Identifiers are integers assigned to each message by the IMAP server that will persist across sessions. They do not change when folders are expunged. Almost all IMAP servers support UIDs.

Each call to the IMAP server can use either message sequence numbers or UIDs in the command arguments and return values. The client specifies to the server which type of identifier should be used. You can set whether IMAPClient should use UIDs or message sequence number via the *use_uid* argument passed when an IMAPClient instance is created and the *use_uid* attribute. The *use_uid* attribute can be used to change the message id type between calls to the server. IMAPClient uses UIDs by default.

Any method that accepts message ids takes either a sequence containing message ids (eg. `[1,2,3]`), or a single message id integer, or a string representing sets and ranges of messages as supported by the IMAP protocol (e.g. `'50-65'`, `'2:*'` or `'2,4:7,9,12:*'`).

3.2.2 Message Flags

An IMAP server keeps zero or more flags for each message. These indicate certain properties of the message or can be used by IMAP clients to keep track of data related to a message.

The IMAPClient package has constants for a number of commmonly used flags:

```
DELETED = br'\Deleted'
SEEN = br'\Seen'
ANSWERED = br'\Answered'
FLAGGED = br'\Flagged'
DRAFT = br'\Draft'
RECENT = br'\Recent'          # This flag is read-only
```

Any method that accepts message flags takes either a sequence containing message flags (eg. `[DELETED, 'foo', 'Bar']`) or a single message flag (eg. `'Foo'`).

3.2.3 Folder Name Encoding

Any method that takes a folder name will accept a standard string or a unicode string. Unicode strings will be transparently encoded using modified UTF-7 as specified by **RFC 3501#section-5.1.3**. This allows for arbitrary unicode characters (eg. non-English characters) to be used in folder names.

The ampersand character ("&") has special meaning in IMAP folder names. IMAPClient automatically escapes and unescapes this character so that the caller doesn't have to.

Automatic folder name encoding and decoding can be enabled or disabled with the *folder_encode* attribute. It defaults to True.

If *folder_encode* is True, all folder names returned by IMAPClient are always returned as unicode strings. If *folder_encode* is False, folder names are returned as str (Python 2) or bytes (Python 3).

3.2.4 Working With Fetched Messages

The IMAP protocol gives access to a limited amount of information about emails stored on the server. In depth analysis of a message usually requires downloading the full message and parsing its content.

The email package of the Python standard library provides a reliable way to transform a raw email into a convenient object.

```python
# Download unread emails and parse them into standard EmailMessage objects
import email

from imapclient import IMAPClient


HOST = 'imap.host.com'
USERNAME = 'someuser'
PASSWORD = 'secret'


with IMAPClient(HOST) as server:
    server.login(USERNAME, PASSWORD)
    server.select_folder('INBOX', readonly=True)

    messages = server.search('UNSEEN')
    for uid, message_data in server.fetch(messages, 'RFC822').items():
        email_message = email.message_from_bytes(message_data[b'RFC822'])
        print(uid, email_message.get('From'), email_message.get('Subject'))
```

3.2.5 TLS/SSL

IMAPClient uses sensible TLS parameter defaults for encrypted connections and also allows for a high level of control of TLS parameters if required. It uses the built-in *ssl* package, provided since Python 2.7.9 and 3.4.

TLS parameters are controlled by passing a `ssl.SSLContext` when creating an IMAPClient instance (or to the *starttls* method when the STARTTLS is used). When `ssl=True` is used without passing a SSLContext, a default context is used. The default context avoids the use of known insecure ciphers and SSL protocol versions, with certificate verification and hostname verification turned on. The default context will use system installed certificate authority trust chains, if available.

When constructing a custom context it is usually best to start with the default context, created by the `ssl` module, and modify it to suit your needs.

> **Warning:** Users of Python 2.7.0 - 2.7.8 can use TLS but cannot configure the settings via an `ssl.SSLContext`. These Python versions are also not capable of proper certification verification. It is highly encouraged to upgrade to a more recent version of Python.

The following example shows how to to disable certification verification and certificate host name checks if required.

```python
# Establish an encrypted connection to a server without checking its
# certificate. This setup is insecure, DO NOT USE to connect to servers
# over the Internet.

import ssl

from imapclient import IMAPClient
```

```python
HOST = 'imap.host.com'
USERNAME = 'someuser'
PASSWORD = 'secret'

ssl_context = ssl.create_default_context()

# don't check if certificate hostname doesn't match target hostname
ssl_context.check_hostname = False

# don't check if the certificate is trusted by a certificate authority
ssl_context.verify_mode = ssl.CERT_NONE

with IMAPClient(HOST, ssl_context=ssl_context) as server:
    server.login(USERNAME, PASSWORD)
    # ...do something...
```

The next example shows how to create a context that will use custom CA certificate. This is required to perform verification of a self-signed certificate used by the IMAP server.

```python
# Establish a secure connection to a server that does not have a certificate
# signed by a trusted certificate authority (CA).

import ssl

from imapclient import IMAPClient

HOST = 'imap.host.com'
USERNAME = 'someuser'
PASSWORD = 'secret'

ssl_context = ssl.create_default_context(cafile="/path/to/cacert.pem")

with IMAPClient(HOST, ssl_context=ssl_context) as server:
    server.login(USERNAME, PASSWORD)
    # ...do something...
```

If your operating system comes with an outdated list of CA certificates you can use the certifi package that provides an up-to-date set of trusted CAs:

```python
import certifi

ssl_context = ssl.create_default_context(cafile=certifi.where())
```

If the server supports it, you can also authenticate using a client certificate:

```python
import ssl

ssl_context = ssl.create_default_context()
ssl_context.load_cert_chain("/path/to/client_certificate.crt")
```

The above examples show some of the most common TLS parameter customisations but there are many other tweaks are possible. Consult the Python 3 `ssl` package documentation for further options.

3.2.6 Logging

IMAPClient logs debug lines using the standard Python `logging` module. Its logger prefix is imapclient..

One way to see debug messages from IMAPClient is to set up logging like this:

```python
import logging

logging.basicConfig(
    format='%(asctime)s - %(levelname)s: %(message)s',
    level=logging.DEBUG
)
```

For advanced usage, please refer to the documentation `logging` module.

3.3 Advanced Usage

This document covers some more advanced features and tips for handling specific usages.

3.3.1 Cleaning Up Connections

To communicate with the server, IMAPClient establishes a TCP connection. It is important for long-lived processes to always close connections at some point to avoid leaking memory and file descriptors. This is usually done with the `logout` method:

```python
import imapclient

c = imapclient.IMAPClient(host="imap.foo.org")
c.login("bar@foo.org", "passwd")
c.select_folder("INBOX")
c.logout()
```

However if an error is raised when selecting the folder, the connection may be left open.

IMAPClient may be used as a context manager that automatically closes connections when they are not needed any more:

```python
import imapclient

with imapclient.IMAPClient(host="imap.foo.org") as c:
    c.login("bar@foo.org", "passwd")
    c.select_folder("INBOX")
```

3.3.2 Watching a Mailbox Using IDLE

The IDLE extension allows an IMAP server to notify a client when something changes in a mailbox. It can be used as an alternative to polling to receive new messages.

The concept is simple: the client connects to the server, selects a mailbox and enters the IDLE mode. At this point the server sends notifications whenever something happens in the selected mailbox until the client ends the IDLE mode by issuing a `DONE` command. This is explained in **RFC 2177**.

```python
# Open a connection in IDLE mode and wait for notifications from the
# server.

from imapclient import IMAPClient
```

```python
HOST = 'imap.host.com'
USERNAME = 'someuser'
PASSWORD = 'password'

server = IMAPClient(HOST)
server.login(USERNAME, PASSWORD)
server.select_folder('INBOX')

# Start IDLE mode
server.idle()
print("Connection is now in IDLE mode, send yourself an email or quit with ^c")

while True:
    try:
        # Wait for up to 30 seconds for an IDLE response
        responses = server.idle_check(timeout=30)
        print("Server sent:", responses if responses else "nothing")
    except KeyboardInterrupt:
        break

server.idle_done()
print("\nIDLE mode done")
server.logout()
```

Note that IMAPClient does not handle low-level socket errors that can happen when maintaining long-lived TCP connections. Users are advised to renew the IDLE command every 10 minutes to avoid the connection from being abruptly closed.

3.3.3 Interactive Sessions

When developing program using IMAPClient is it sometimes useful to have an interactive shell to play with. IMAPClient ships with a module that lets you fire up an interactive shell with an IMAPClient instance connected to an IMAP server.

Start a session like this:

```
python -m imapclient.interact -H <host> -u <user> ...
```

Various options are available to specify the IMAP server details. See the help (–help) for more details. You'll be prompted for a username and password if one isn't provided on the command line.

It is also possible to pass connection details as a configuration file like this:

```
python -m imapclient.interact -f <config file>
```

See below for details of the *configuration file format*.

If installed, IPython will be used as the embedded shell. Otherwise the basic built-in Python shell will be used.

The connected IMAPClient instance is available as the variable "c". Here's an example session:

```
$ python -m imapclient.interact -H <host> -u <user> ...
Connecting...
Connected.

IMAPClient instance is "c"
```

```
In [1]: c.select_folder('inbox')
Out[1]:
{b'EXISTS': 2,
 b'FLAGS': (b'\\Answered',
    b'\\Flagged',
    b'\\Deleted',
    b'\\Seen',
    b'\\Draft'),
 b'PERMANENTFLAGS': (b'\\Answered',
    b'\\Flagged',
    b'\\Deleted',
    b'\\Seen',
    b'\\Draft'),
 b'READ-WRITE': True,
 b'RECENT': 0,
 b'UIDNEXT': 1339,
 b'UIDVALIDITY': 1239278212}

In [2]: c.search()
Out[2]: [1123, 1233]

In [3]: c.logout()
Out[3]: b'Logging out'
```

Configuration File Format

Both the IMAPClient interactive shell and the live tests take configuration files which specify how to to connect to an IMAP server. The configuration file format is the same for both.

Configuration files use the INI format and must always have a section called `DEFAULT`. Here's a simple example:

```
[DEFAULT]
host = imap.mailserver.com
username = bob
password = sekret
ssl = True
```

The supported options are:

Name	Type	Description
host	string	IMAP hostname to connect to.
username	string	The username to authenticate as.
password	string	The password to use with `username`.
port	int	Server port to connect to. Defaults to 143 unless `ssl` is True.
ssl	bool	Use SSL/TLS to connect.
starttls	bool	Use STARTTLS to connect.
ssl_check_hostname	bool	If true and SSL is in use, check that certificate matches the hostname (defaults to true)
ssl_verify_cert	bool	If true and SSL is in use, check that the certifcate is valid (defaults to true).
ssl_ca_file	string	If SSL is true, use this to specify certificate authority certs to validate with.
timeout	int	Time out I/O operations after this many seconds.
oauth2	bool	If true, use OAUTH2 to authenticate (`username` and `password` are ignored).
oauth2_client_id	string	OAUTH2 client id.
oauth2_client_secret	string	OAUTH2 client secret.
oauth2_refresh_token	string	OAUTH2 token for refreshing the secret.

Acceptable boolean values are "1", "yes", "true", and "on", for true; and "0", "no", "false", and "off", for false.

API Reference

This section describes public functions and classes of IMAPClient library.

4.1 IMAPClient Class

The primary class used by the imapclient package is the IMAPClient class. All interaction with a remote IMAP server is performed via an IMAPClient instance.

class imapclient.**IMAPClient**(*host*, *port=None*, *use_uid=True*, *ssl=True*, *stream=False*, *ssl_context=None*, *timeout=None*)

A connection to the IMAP server specified by *host* is made when this class is instantiated.

port defaults to 993, or 143 if *ssl* is `False`.

If *use_uid* is `True` unique message UIDs be used for all calls that accept message ids (defaults to `True`).

If *ssl* is `True` (the default) a secure connection will be made. Otherwise an insecure connection over plain text will be established.

If *ssl* is `True` the optional *ssl_context* argument can be used to provide an `ssl.SSLContext` instance used to control SSL/TLS connection parameters. If this is not provided a sensible default context will be used.

If *stream* is `True` then *host* is used as the command to run to establish a connection to the IMAP server (defaults to `False`). This is useful for exotic connection or authentication setups.

Use *timeout* to specify a timeout for the socket connected to the IMAP server. The timeout can be either a float number, or an instance of *imapclient.SocketTimeout*.

- If a single float number is passed, the same timeout delay applies during the initial connection to the server and for all future socket reads and writes.

- In case of a `SocketTimeout`, connection timeout and read/write operations can have distinct timeouts.

- The default is `None`, where no timeout is used.

The *normalise_times* attribute specifies whether datetimes returned by `fetch()` are normalised to the local system time and include no timezone information (native), or are datetimes that include timezone information

(aware). By default *normalise_times* is True (times are normalised to the local system time). This attribute can be changed between `fetch()` calls if required.

Can be used as a context manager to automatically close opened connections:

```
>>> with IMAPClient(host="imap.foo.org") as client:
...     client.login("bar@foo.org", "passwd")
```

AbortError
    alias of `imaplib.abort`

Error
    alias of `imaplib.error`

ReadOnlyError
    alias of `imaplib.readonly`

add_flags(*messages*, *flags*, *silent=False*)
    Add *flags* to *messages* in the currently selected folder.

    flags should be a sequence of strings.

    Returns the flags set for each modified message (see *get_flags*), or None if *silent* is true.

add_gmail_labels(*messages*, *labels*, *silent=False*)
    Add *labels* to *messages* in the currently selected folder.

    labels should be a sequence of strings.

    Returns the label set for each modified message (see *get_gmail_labels*), or None if *silent* is true.

    This only works with IMAP servers that support the X-GM-LABELS attribute (eg. Gmail).

append(*folder*, *msg*, *flags=()*, *msg_time=None*)
    Append a message to *folder*.

    msg should be a string contains the full message including headers.

    flags should be a sequence of message flags to set. If not specified no flags will be set.

    msg_time is an optional datetime instance specifying the date and time to set on the message. The server will set a time if it isn't specified. If *msg_time* contains timezone information (tzinfo), this will be honoured. Otherwise the local machine's time zone sent to the server.

    Returns the APPEND response as returned by the server.

capabilities()
    Returns the server capability list.

    If the session is authenticated and the server has returned an untagged CAPABILITY response at authentication time, this response will be returned. Otherwise, the CAPABILITY command will be issued to the server, with the results cached for future calls.

    If the session is not yet authenticated, the capabilities requested at connection time will be returned.

close_folder()
    Close the currently selected folder, returning the server response string.

copy(*messages*, *folder*)
    Copy one or more messages from the current folder to *folder*. Returns the COPY response string returned by the server.

create_folder(*folder*)
    Create *folder* on the server returning the server response string.

delete_folder(*folder*)
> Delete *folder* on the server returning the server response string.

delete_messages(*messages*, *silent=False*)
> Delete one or more *messages* from the currently selected folder.
>
> Returns the flags set for each modified message (see *get_flags*).

enable(**args*, ***kwargs*)
> Activate one or more server side capability extensions.
>
> Most capabilities do not need to be enabled. This is only required for extensions which introduce backwards incompatible behaviour. Two capabilities which may require enable are `CONDSTORE` and `UTF8=ACCEPT`.
>
> A list of the requested extensions that were successfully enabled on the server is returned.
>
> Once enabled each extension remains active until the IMAP connection is closed.
>
> See **RFC 5161** for more details.

expunge(*messages=None*)
> When, no *messages* are specified, remove all messages from the currently selected folder that have the `\Deleted` flag set.
>
> The return value is the server response message followed by a list of expunge responses. For example:

```
('Expunge completed.',
 [(2, 'EXPUNGE'),
  (1, 'EXPUNGE'),
  (0, 'RECENT')])
```

> In this case, the responses indicate that the message with sequence numbers 2 and 1 where deleted, leaving no recent messages in the folder.
>
> See **RFC 3501#section-6.4.3** section 6.4.3 and **RFC 3501#section-7.4.1** section 7.4.1 for more details.
>
> When *messages* are specified, remove the specified messages from the selected folder, provided those messages also have the `\Deleted` flag set. The return value is `None` in this case.
>
> Expunging messages by id(s) requires that *use_uid* is `True` for the client.
>
> See **RFC 4315#section-2.1** section 2.1 for more details.

fetch(*messages*, *data*, *modifiers=None*)
> Retrieve selected *data* associated with one or more *messages* in the currently selected folder.
>
> *data* should be specified as a sequnce of strings, one item per data selector, for example `['INTERNALDATE', 'RFC822']`.
>
> *modifiers* are required for some extensions to the IMAP protocol (eg. **RFC 4551**). These should be a sequnce of strings if specified, for example `['CHANGEDSINCE 123']`.
>
> A dictionary is returned, indexed by message number. Each item in this dictionary is also a dictionary, with an entry corresponding to each item in *data*. Returned values will be appropriately typed. For example, integer values will be returned as Python integers, timestamps will be returned as datetime instances and ENVELOPE responses will be returned as *Envelope* instances.
>
> String data will generally be returned as bytes (Python 3) or str (Python 2).
>
> In addition to an element for each *data* item, the dict returned for each message also contains a *SEQ* key containing the sequence number for the message. This allows for mapping between the UID and sequence number (when the *use_uid* property is `True`).

Example:

```
>> c.fetch([3293, 3230], ['INTERNALDATE', 'FLAGS'])
{3230: {b'FLAGS': (b'\Seen',),
        b'INTERNALDATE': datetime.datetime(2011, 1, 30, 13, 32, 9),
        b'SEQ': 84},
 3293: {b'FLAGS': (),
        b'INTERNALDATE': datetime.datetime(2011, 2, 24, 19, 30, 36),
        b'SEQ': 110}}
```

find_special_folder(*folder_flag*)
    Try to locate a special folder, like the Sent or Trash folder.

```
>>> server.find_special_folder(imapclient.SENT)
'INBOX.Sent'
```

This function tries its best to find the correct folder (if any) but uses heuristics when the server is unable to precisely tell where special folders are located.

Returns the name of the folder if found, or None otherwise.

folder_exists(*folder*)
    Return `True` if *folder* exists on the server.

folder_status(*folder*, *what=None*)
    Return the status of *folder*.

    what should be a sequence of status items to query. This defaults to (`'MESSAGES'`, `'RECENT'`, `'UIDNEXT'`, `'UIDVALIDITY'`, `'UNSEEN'`).

    Returns a dictionary of the status items for the folder with keys matching *what*.

get_flags(*messages*)
    Return the flags set for each message in *messages* from the currently selected folder.

    The return value is a dictionary structured like this: `{ msgid1:  (flag1, flag2, ... ), }`.

get_gmail_labels(*messages*)
    Return the label set for each message in *messages* in the currently selected folder.

    The return value is a dictionary structured like this: `{ msgid1:  (label1, label2, ... ), }`.

    This only works with IMAP servers that support the X-GM-LABELS attribute (eg. Gmail).

get_quota(**args*, ***kwargs*)
    Get the quotas associated with a mailbox.

    Returns a list of Quota objects.

get_quota_root(**args*, ***kwargs*)
    Get the quota roots for a mailbox.

    The IMAP server responds with the quota root and the quotas associated so there is usually no need to call *get_quota* after.

    See **RFC 2087** for more details.

    Return a tuple of MailboxQuotaRoots and list of Quota associated

getacl(**args*, ***kwargs*)
    Returns a list of (`who,  acl`) tuples describing the access controls for *folder*.

gmail_search(**args*, ***kwargs*)
    Search using Gmail's X-GM-RAW attribute.

---

query should be a valid Gmail search query string. For example: `has:attachment in:unread`. The search string may be unicode and will be encoded using the specified *charset* (defaulting to UTF-8).

This method only works for IMAP servers that support X-GM-RAW, which is only likely to be Gmail.

See https://developers.google.com/gmail/imap_extensions#extension_of_the_search_command_x-gm-raw for more info.

has_capability(*capability*)
    Return `True` if the IMAP server has the given *capability*.

id_(**args*, ***kwargs*)
    Issue the ID command, returning a dict of server implementation fields.

    parameters should be specified as a dictionary of field/value pairs, for example: `{"name": "IMAPClient", "version": "0.12"}`

idle(**args*, ***kwargs*)
    Put the server into IDLE mode.

    In this mode the server will return unsolicited responses about changes to the selected mailbox. This method returns immediately. Use `idle_check()` to look for IDLE responses and `idle_done()` to stop IDLE mode.

    Note: Any other commmands issued while the server is in IDLE mode will fail.

    See **RFC 2177** for more information about the IDLE extension.

idle_check(**args*, ***kwargs*)
    Check for any IDLE responses sent by the server.

    This method should only be called if the server is in IDLE mode (see `idle()`).

    By default, this method will block until an IDLE response is received. If *timeout* is provided, the call will block for at most this many seconds while waiting for an IDLE response.

    The return value is a list of received IDLE responses. These will be parsed with values converted to appropriate types. For example:

```
[(b'OK', b'Still here'),
 (1, b'EXISTS'),
 (1, b'FETCH', (b'FLAGS', (b'\NotJunk',)))]
```

idle_done(**args*, ***kwargs*)
    Take the server out of IDLE mode.

    This method should only be called if the server is already in IDLE mode.

    The return value is of the form (`command_text`, `idle_responses`) where *command_text* is the text sent by the server when the IDLE command finished (eg. `b'Idle terminated'`) and *idle_responses* is a list of parsed idle responses received since the last call to `idle_check()` (if any). These are returned in parsed form as per `idle_check()`.

list_folders(*directory=u''*, *pattern=u'*'*)
    Get a listing of folders on the server as a list of (`flags`, `delimiter`, `name`) tuples.

    Specifying *directory* will limit returned folders to the given base directory. The directory and any child directories will returned.

Specifying *pattern* will limit returned folders to those with matching names. The wildcards are supported in *pattern*. `*` matches zero or more of any character and `%` matches 0 or more characters except the folder delimiter.

Calling list_folders with no arguments will recursively list all folders available for the logged in user.

Folder names are always returned as unicode strings, and decoded from modified UTF-7, except if folder_decode is not set.

list_sub_folders(*directory=u''*, *pattern=u'*'*)
Return a list of subscribed folders on the server as (`flags`, `delimiter`, `name`) tuples.

The default behaviour will list all subscribed folders. The *directory* and *pattern* arguments are as per list_folders().

login(*username*, *password*)
Login using *username* and *password*, returning the server response.

logout()
Logout, returning the server response.

move(**args*, ***kwargs*)
Atomically move messages to another folder.

Requires the MOVE capability, see **RFC 6851**.

> **Parameters**
>
> - **messages** – List of message UIDs to move.
>
> - **folder** – The destination folder name.

namespace(**args*, ***kwargs*)
Return the namespace for the account as a (personal, other, shared) tuple.

Each element may be None if no namespace of that type exists, or a sequence of (prefix, separator) pairs.

For convenience the tuple elements may be accessed positionally or using attributes named *personal*, *other* and *shared*.

See **RFC 2342** for more details.

noop()
Execute the NOOP command.

This command returns immediately, returning any server side status updates. It can also be used to reset any auto-logout timers.

The return value is the server command response message followed by a list of status responses. For example:

```
(b'NOOP completed.',
 [(4, b'EXISTS'),
  (3, b'FETCH', (b'FLAGS', (b'bar', b'sne'))),
  (6, b'FETCH', (b'FLAGS', (b'sne',)))])
```

oauth2_login(*user*, *access_token*, *mech=u'XOAUTH2'*, *vendor=None*)
Authenticate using the OAUTH2 method.

Gmail and Yahoo both support the 'XOAUTH2' mechanism, but Yahoo requires the 'vendor' portion in the payload.

plain_login(*identity*, *password*, *authorization_identity=None*)
Authenticate using the PLAIN method (requires server support).

remove_flags(*messages*, *flags*, *silent=False*)

Remove one or more *flags* from *messages* in the currently selected folder.

flags should be a sequence of strings.

Returns the flags set for each modified message (see *get_flags*), or None if *silent* is true.

remove_gmail_labels(*messages*, *labels*, *silent=False*)

Remove one or more *labels* from *messages* in the currently selected folder, or None if *silent* is true.

labels should be a sequence of strings.

Returns the label set for each modified message (see *get_gmail_labels*).

This only works with IMAP servers that support the X-GM-LABELS attribute (eg. Gmail).

rename_folder(*old_name*, *new_name*)

Change the name of a folder on the server.

search(*criteria=u'ALL'*, *charset=None*)

Return a list of messages ids from the currently selected folder matching *criteria*.

criteria should be a sequence of one or more criteria items. Each criteria item may be either unicode or bytes. Example values:

```
[u'UNSEEN']
[u'SMALLER', 500]
[b'NOT', b'DELETED']
[u'TEXT', u'foo bar', u'FLAGGED', u'SUBJECT', u'baz']
[u'SINCE', date(2005, 4, 3)]
```

IMAPClient will perform conversion and quoting as required. The caller shouldn't do this.

It is also possible (but not recommended) to pass the combined criteria as a single string. In this case IMAPClient won't perform quoting, allowing lower-level specification of criteria. Examples of this style:

```
u'UNSEEN'
u'SMALLER 500'
b'NOT DELETED'
u'TEXT "foo bar" FLAGGED SUBJECT "baz"'
b'SINCE 03-Apr-2005'
```

To support complex search expressions, criteria lists can be nested. IMAPClient will insert parentheses in the right places. The following will match messages that are both not flagged and do not have "foo" in the subject:

['NOT', ['SUBJECT', 'foo', 'FLAGGED']]

charset specifies the character set of the criteria. It defaults to US-ASCII as this is the only charset that a server is required to support by the RFC. UTF-8 is commonly supported however.

Any criteria specified using unicode will be encoded as per *charset*. Specifying a unicode criteria that can not be encoded using *charset* will result in an error.

Any criteria specified using bytes will be sent as-is but should use an encoding that matches *charset* (the character set given is still passed on to the server).

See **RFC 3501#section-6.4.4** for more details.

Note that criteria arguments that are 8-bit will be transparently sent by IMAPClient as IMAP literals to ensure adherence to IMAP standards.

The returned list of message ids will have a special *modseq* attribute. This is set if the server included a MODSEQ value to the search response (i.e. if a MODSEQ criteria was included in the search).

---

select_folder(*folder*, *readonly=False*)
> Set the current folder on the server.
>
> Future calls to methods such as search and fetch will act on the selected folder.
>
> Returns a dictionary containing the SELECT response. At least the b'EXISTS', b'FLAGS' and b'RECENT' keys are guaranteed to exist. An example:

```
{b'EXISTS': 3,
 b'FLAGS': (b'\Answered', b'\Flagged', b'\Deleted', ... ),
 b'RECENT': 0,
 b'PERMANENTFLAGS': (b'\Answered', b'\Flagged', b'\Deleted', ... ),
 b'READ-WRITE': True,
 b'UIDNEXT': 11,
 b'UIDVALIDITY': 1239278212}
```

set_flags(*messages*, *flags*, *silent=False*)
> Set the *flags* for *messages* in the currently selected folder.
>
> *flags* should be a sequence of strings.
>
> Returns the flags set for each modified message (see *get_flags*), or None if *silent* is true.

set_gmail_labels(*messages*, *labels*, *silent=False*)
> Set the *labels* for *messages* in the currently selected folder.
>
> *labels* should be a sequence of strings.
>
> Returns the label set for each modified message (see *get_gmail_labels*), or None if *silent* is true.
>
> This only works with IMAP servers that support the X-GM-LABELS attribute (eg. Gmail).

set_quota(**args*, ***kwargs*)
> Set one or more quotas on resources.
>
> > **Parameters quotas** – list of Quota objects

setacl(**args*, ***kwargs*)
> Set an ACL (*what*) for user (*who*) for a folder.
>
> Set *what* to an empty string to remove an ACL. Returns the server response string.

shutdown()
> Close the connection to the IMAP server (without logging out)
>
> In most cases, *logout()* should be used instead of this. The logout method also shutdown down the connection.

sort(**args*, ***kwargs*)
> Return a list of message ids from the currently selected folder, sorted by *sort_criteria* and optionally filtered by *criteria*.
>
> *sort_criteria* may be specified as a sequence of strings or a single string. IMAPClient will take care any required conversions. Valid *sort_criteria* values:

```
['ARRIVAL']
['SUBJECT', 'ARRIVAL']
'ARRIVAL'
'REVERSE SIZE'
```

> The *criteria* and *charset* arguments are as per *search()*.
>
> See **RFC 5256** for full details.

Note that SORT is an extension to the IMAP4 standard so it may not be supported by all IMAP servers.

starttls(**args*, ***kwargs*)

Switch to an SSL encrypted connection by sending a STARTTLS command.

The *ssl_context* argument is optional and should be a `ssl.SSLContext` object. If no SSL context is given, a SSL context with reasonable default settings will be used.

You can enable checking of the hostname in the certificate presented by the server against the hostname which was used for connecting, by setting the *check_hostname* attribute of the SSL context to `True`. The default SSL context has this setting enabled.

Raises *Error* if the SSL connection could not be established.

Raises *AbortError* if the server does not support STARTTLS or an SSL connection is already established.

subscribe_folder(*folder*)

Subscribe to *folder*, returning the server response string.

thread(*algorithm=u'REFERENCES'*, *criteria=u'ALL'*, *charset=u'UTF-8'*)

Return a list of messages threads from the currently selected folder which match *criteria*.

Each returned thread is a list of messages ids. An example return value containing three message threads:

```
((1, 2), (3,), (4, 5, 6))
```

The optional *algorithm* argument specifies the threading algorithm to use.

The *criteria* and *charset* arguments are as per *search()*.

See **RFC 5256** for more details.

unselect_folder(**args*, ***kwargs*)

Unselect the current folder and release associated resources.

Unlike `close_folder`, the `UNSELECT` command does not expunge the mailbox, keeping messages with Deleted flag set for example.

Returns the UNSELECT response string returned by the server.

unsubscribe_folder(*folder*)

Unsubscribe to *folder*, returning the server response string.

welcome

access the server greeting message

xlist_folders(**args*, ***kwargs*)

Execute the XLIST command, returning (`flags, delimiter, name`) tuples.

This method returns special flags for each folder and a localized name for certain folders (e.g. the name of the inbox may be localized and the flags can be used to determine the actual inbox, even if the name has been localized.

A `XLIST` response could look something like:

```
[((b'\HasNoChildren', b'\Inbox'), b'/', u'Inbox'),
 ((b'\Noselect', b'\HasChildren'), b'/', u'[Gmail]'),
 ((b'\HasNoChildren', b'\AllMail'), b'/', u'[Gmail]/All Mail'),
 ((b'\HasNoChildren', b'\Drafts'), b'/', u'[Gmail]/Drafts'),
 ((b'\HasNoChildren', b'\Important'), b'/', u'[Gmail]/Important'),
 ((b'\HasNoChildren', b'\Sent'), b'/', u'[Gmail]/Sent Mail'),
 ((b'\HasNoChildren', b'\Spam'), b'/', u'[Gmail]/Spam'),
```

```
 ((b'\HasNoChildren', b'\Starred'), b'/', u'[Gmail]/Starred'),
 ((b'\HasNoChildren', b'\Trash'), b'/', u'[Gmail]/Trash')]
```

This is a *deprecated* Gmail-specific IMAP extension (See https://developers.google.com/gmail/imap_extensions#xlist_is_deprecated for more information).

The *directory* and *pattern* arguments are as per list_folders().

class imapclient.**SocketTimeout**

Represents timeout configuration for an IMAP connection.

Variables

- **connect** – maximum time to wait for a connection attempt to remote server

- **read** – maximum time to wait for performing a read/write operation

As an example, SocketTimeout(connect=15, read=60) will make the socket timeout if the connection takes more than 15 seconds to establish but read/write operations can take up to 60 seconds once the connection is done.

4.2 Fetch Response Types

Various types may be used in the data structures returned by *IMAPClient.fetch()* when certain response types are encountered during parsing.

class imapclient.response_types.**Address**

Represents electronic mail addresses. Used to store addresses in *Envelope*.

Variables

- **name** – The address "personal name".

- **route** – SMTP source route (rarely used).

- **mailbox** – Mailbox name (what comes just before the @ sign).

- **host** – The host/domain name.

As an example, an address header that looks like:

```
Mary Smith <mary@foo.com>
```

would be represented as:

```
Address(name=u'Mary Smith', route=None, mailbox=u'mary', host=u'foo.com')
```

See **RFC 2822** for more detail.

See also *Envelope* for information about handling of "group syntax".

class imapclient.response_types.**BodyData**

Returned when parsing BODY and BODYSTRUCTURE responses.

class imapclient.response_types.**Envelope**

Represents envelope structures of messages. Returned when parsing ENVELOPE responses.

Variables

- **date** – A datetime instance that represents the "Date" header.

- **subject** – A string that contains the "Subject" header.

- **from_** – A tuple of Address objects that represent one or more addresses from the "From" header, or None if header does not exist.

- **sender** – As for from_ but represents the "Sender" header.

- **reply_to** – As for from_ but represents the "Reply-To" header.

- **to** – As for from_ but represents the "To" header.

- **cc** – As for from_ but represents the "Cc" header.

- **bcc** – As for from_ but represents the "Bcc" recipients.

- **in_reply_to** – A string that contains the "In-Reply-To" header.

- **message_id** – A string that contains the "Message-Id" header.

A particular issue to watch out for is IMAP's handling of "group syntax" in address fields. This is often encountered as a recipient header of the form:

```
undisclosed-recipients:;
```

but can also be expressed per this more general example:

```
A group: a@example.com, B <b@example.org>;
```

This example would yield the following Address tuples:

```
Address(name=None, route=None, mailbox=u'A group', host=None)
Address(name=None, route=None, mailbox=u'a', host=u'example.com')
Address(name=u'B', route=None, mailbox=u'b', host=u'example.org')
Address(name=None, route=None, mailbox=None, host=None)
```

The first Address, where host is None, indicates the start of the group. The mailbox field contains the group name. The final Address, where both mailbox and host are None, indicates the end of the group.

See **RFC 3501#section-7.4.2** and **RFC 2822** for further details.

class imapclient.response_types.**SearchIds**(**args*)
    Contains a list of message ids as returned by IMAPClient.search().

    The *modseq* attribute will contain the MODSEQ value returned by the server (only if the SEARCH command sent involved the MODSEQ criteria). See **RFC 4551** for more details.

4.3 Exceptions

IMAPClient wraps exceptions raised by imaplib to ease the error handling. All the exceptions related to IMAP errors are defined in the module *imapclient.exceptions*. The following general exceptions may be raised:

- IMAPClientError: the base class for IMAPClient's exceptions and the most commonly used error.

- IMAPClientAbortError: raised if a serious error has occurred that means the IMAP connection is no longer usable. The connection should be dropped without logout if this occurs.

- IMAPClientReadOnlyError: raised if a modifying operation was attempted on a read-only folder.

More specific exceptions existed for common known errors:

exception imapclient.exceptions.**CapabilityError**
    The command tried by the user needs a capability not installed on the IMAP server

exception `imapclient.exceptions.`**`IllegalStateError`**
  The command tried needs a different state to be executed. This means the user is not logged in or the command needs a folder to be selected.

exception `imapclient.exceptions.`**`InvalidCriteriaError`**
  A command using a search criteria failed, probably due to a syntax error in the criteria string.

exception `imapclient.exceptions.`**`LoginError`**
  A connection has been established with the server but an error occurred during the authentication.

exception `imapclient.exceptions.`**`ProtocolError`**
  The server replied with a response that violates the IMAP protocol.

Exceptions from lower layers are possible, such as networks error or unicode malformed exception. In particular:

- socket.error

- socket.timeout: raised if a timeout was specified when creating the IMAPClient instance and a network operation takes too long.

- ssl.SSLError: the base class for network or SSL protocol errors when `ssl=True` or `starttls()` is used.

- ssl.CertificateError: raised when TLS certification verification fails. This is *not* a subclass of SSLError.

4.4 Utilities

class `imapclient.testable_imapclient.`**`MockIMAP4`**(**args*, ***kwargs*)

class `imapclient.testable_imapclient.`**`TestableIMAPClient`**
  Wrapper of *`imapclient.IMAPClient`* that mocks all interaction with real IMAP server.

  This class should only be used in tests, where you can safely interact with imapclient without running commands on a real IMAP account.

4.5 TLS Support

This module contains IMAPClient's functionality related to Transport Layer Security (TLS a.k.a. SSL).

class `imapclient.tls.`**`IMAP4_TLS`**(*host*, *port*, *ssl_context*, *timeout*)
  IMAP4 client class for TLS/SSL connections.

  Adapted from imaplib.IMAP4_SSL.

4.6 Thread Safety

Instances of IMAPClient are NOT thread safe. They should not be shared and accessed concurrently from multiple threads.

CHAPTER 5

Contributor Guide

5.1 Contributing to IMAPClient

The best way to contribute changes to IMAPClient is to fork the official repository on Github, make changes in a branch in your personal fork and then submit a pull request.

Discussion on the mailing list before undertaking development is highly encouraged for potentially major changes.

Although not essential, it will make the project maintainers much happier if change submissions include appropriate updates to unit tests, live tests and documentation. Please ask if you're unsure how of how the tests work.

Please read on if you plan on submitting changes to IMAPClient.

5.1.1 Source Code

The official source code repository for IMAPClient can be found on Github at: https://github.com/mjs/imapclient

Any major feature work will also be found as branches of this repository.

Branches

Development for the next major release happens on the `master` branch.

There is also a branch for each major release series (for example: `1.x`). When appropriate and when there will be future releases for a series, changes may be selectively merged between `master` and a stable release branch.

Release Tags

Each released version is available in the IMAPClient repository as a Git tag (e.g. "0.9.1").

5.1.2 Unit Tests

Running Unit Tests

To run the tests, from the root of the package source run:

```
python setup.py test
```

Testing Against Multiple Python Versions

When submitting a Pull Request to IMAPClient, tests are automatically ran against all the supported Python versions.

It is possible to locally run these tests using tox. Once installed, the `tox` command will use the tox.ini file in the root of the project source and run the unit tests against the Python versions officially supported by IMAPClient (provided these versions of Python are installed!).

Writing Unit Tests

Protocol level unit tests should not act against a real IMAP server but should use canned data instead. The IMAP-ClientTest base class should typically be used as the base class for any tests - it provides a mock IMAPClient instance at *self.client*. See the tests in *tests/test_imapclient.py* for examples of how to write unit tests using this approach.

5.1.3 Documentation

The source for the project's documentation can be found under doc/src in the source distribution.

In order to build the documentation you'll need install Sphinx. Running `pip install '.[doc]'` from the root of the project source will do this.

Once Sphinx is installed, the documentation can be rebuilt using:

```
python setup.py build_sphinx
```

External Documentation

The Unofficial IMAP Protocol Wiki is very useful when writing IMAP related software and is highly recommended.

Authors

IMAPClient was created by Menno Finlay-Smits <inbox@menno.io>. The project is now maintained by Nicolas Le Manchet and Menno Finlay-Smits.

Many thanks go to the following people for their help with this project:

- Maxime Lorant
- Mathieu Agopian
- Chris Arndt
- Jp Calderone
- John Louis del Rosario
- Dave Eckhardt
- Eben Freeman
- Helder Guerreiro
- Mark Hammond
- Johannes Heckel
- Thomas Jost
- Lukasz Mierzwa
- Naveen Nathan
- Brian Neal
- Phil Peterson
- Aviv Salem
- Andrew Scheller
- Thomas Steinacher
- Zac Witte

- Hans-Peter Jansen

Release History

8.1 Version 2.1.0

8.1.1 Changed

- TravisCI now runs tests against PyPy

- Python 3.7 is now officially supported

- Cleaned up server capability checks

- Use TLS by default for interactive sessions

8.1.2 Added

- Support the `QUOTA` extension

- Support for locating special folders (`find_special_folder()`)

- Document usage of client TLS certificates

- Added documentation & example for parsing retrieved emails using the standard library `email` package.

8.1.3 Fixed

- Handle `NIL` values for `INTERNALDATE`

8.2 Version 2.0

8.2.1 Changed

- Only use Python's built-in TLS support (no more backports.ssl & pyOpenSSL)
- Connections use SSL/TLS by default (*ssl=True*)
- Drop `imapclient.tls.create_default_context` function. In case you were using it, you can use the method with the same name available in the built-in `ssl` module.
- Logs are now handled by the Python logging module. The *debug* and *log_file* attributes are gone.
- More precise exceptions available in *imapclient.exceptions* are raised when an error happens
- *imapclient.exceptions.ProtocolError* is now raised when the reply from a remote server violates the IMAP protocol.
- SEARCH exceptions now link to relevant documentation.
- GMail labels are now strings instead of bytes in Python 3.
- OAUTH v1 support removed.
- setup.py has been simplified.
- All non-library code moved out of the *imapclient* package.
- Many documentation improvements.

8.2.2 Added

- Connection and read/write operations timeout can now be distinct, using *imapclient.SocketTimeout* namedtuple as *timeout* parameter.
- A context manager is introduced to automatically close connections to remote servers.
- EXPUNGE by ID support.
- ENABLE support.
- UNSELECT support.
- Atomically move messages to another folder using the MOVE extension (**RFC 6851**)
- New *welcome* property to allow access to IMAP server greeting.

8.2.3 Fixed

- GMail labels using international characters are now handled properly.
- Don't use locale dependent formatting in *datetime_to_INTERNAL_DATE()*.
- Quote empty strings to prevent syntax errors while SEARCHing for zero-length strings.
- Handle address without mailbox name or host in Address namedtuple.
- Avoid asserts in response parsing codes to allow graceful recovery.
- Prevent logging of IMAP passwords.

8.2.4 Python compatibility

Support for Python 2.6 and 3.3 is removed in this release.

This version supports Python 2.7, 3.4, 3.5 and 3.6. We officially support the latest release of each series.

8.3 Version 1.1.0

8.3.1 Added

- Search now supports nested criteria so that more complex criteria can be expressed. IMAPClient will add parentheses in the right place.
- PLAIN authentication support (via *plain_login* method)
- *unselect_folder()* method, for servers with the UNSELECT capability (#200)
- Add ENABLE support (#136)
- UID EXPUNGE support (#287)

8.3.2 Changed

- the *mock* package is no longer installed by default (just as a test dependency)
- handle NIL date values in INTERNALDATE
- add *silent* option to all flags methods (improves performance by avoiding unnecessary parsing)
- simplify Gmail label functionality
- folder_status is more robust
- various livetest reliability improvements

8.3.3 Fixed

- don't quote search criteria when sent as IMAP literals. Fixes #249.
- Modified UTF-7 encoding function had quirks in its original algorithm, leading to incorrect encoded output in some cases. The algorithm, described in RFC 3501, has been reimplemented to fix #187 and is better documented.
- use fixed month names when formatting INTERNALDATES (don't rely on locale)
- handle address without mailbox name or host in Address namedtuple. Fixes #242.
- Use cryptography < 2.0 on Python 3.3. Fixes #305.

8.4 Version 1.0.2

8.4.1 New

- Documented the livetest/interact INI file format.
- Documented handling of RFC2822 group syntax.

8.4.2 Changed

- Explicitly check that the required pyOpenSSL version is installed
- Start testing against Python 3.5
- Update doc links from readthedocs.org to readthedocs.io
- Rearranged README so that project essentials are right at the top.

8.4.3 Fixed

- Allow installation from alternate directories

8.5 Version 1.0.1

8.5.1 Changed

- Minimum backports.ssl dependency is now 0.0.9 (an important performance issue was addressed)
- setuptools 18.8.1 now used due to strange zip file error for 17.1

8.5.2 Fixed

- Unit test for version strings were updated to now always include the patch version.
- Fresh capabilities now retrieved between STARTTLS and authentication (#195).

8.6 Version 1.0.0

8.6.1 Enhanced TLS support [API]

The way that IMAPClient establishes TLS/SSL connections has been completely reworked. By default IMAPClient will attempt certificate verification, certificate hostname checking, and will not use known-insecure TLS settings and protocols. In addition, TLS parameters are now highly configurable.

By leveraging pyOpenSSL and backports.ssl, all Python versions supported by IMAPClient enjoy the same TLS functionality and API.

These packages mean that IMAPClient now has a number of new dependencies. These should be installed automatically as required but there will no doubt be complications.

Compatibility breaks:

1. Due to lack of support in some of the dependent libraries, IMAPClient no longer supports Python 3.2.
2. The passthrough keyword arguments that the IMAPClient constructor took in past versions are no longer accepted. These were in place to provide access to imaplib's SSL arguments which are no longer relevant. Please pass a SSL context object instead.
3. When using the default SSL context that IMAPClient creates (recommended), certificate verification is enabled. This means that IMAPClient connections to servers that used to work before, may fail now (especially if a self-signed certificate is used by the server). Refer to the documentation for details of how to supply alternate CA certificates or disable verification.

4. There are some new exceptions that might be raised in response to network issues or TLS protocol failures. Refer to the Exceptions section of the manual for more details.

Please refer to the "TLS/SSL" section of the manual for more details on all of the above.

Many thanks to Chris Arndt and Marc-Antoine Parent for their input into these TLS improvements.

8.6.2 STARTTLS support [NEW]

When the server supports it, IMAPClient can now establish an encrypted connection after initially starting with an un-encrypted connection using the STARTTLS command. The starttls method takes an SSL context object for controlling the parameters of the TLS negotiation.

Many thanks to Chris Arndt for his extensive initial work on this.

8.6.3 More robust criteria handling for search, sort and thread [API]

IMAPClient's methods that accept search criteria (search, sort, thread, gmail_search) have been changed to provide take criteria in a more straightforward and robust way. In addition, the way the *charset* argument interacts with search criteria has been improved. These changes make it easier to pass search criteria and have them handled correctly but unfortunately also mean that small changes may be required to existing code that uses IMAPClient.

Search criteria

The preferred way to specify criteria now is as a list of strings, ints and dates (where relevant). The list should be flat with all the criteria parts run together. Where a criteria takes an argument, just provide it as the next element in the list.

Some valid examples:

```
c.search(['DELETED'])
c.search(['NOT', 'DELETED'])
c.search(['FLAGGED', 'SUBJECT', 'foo', 'BODY', 'hello world'])
c.search(['NOT', 'DELETED', 'SMALLER', 1000])
c.search(['SINCE', date(2006, 5, 3)])
```

IMAPClient will perform all required conversion, quoting and encoding. Callers do not need to and should not attempt to do this themselves. IMAPClient will automatically send criteria parts as IMAP literals when required (i.e. when the encoded part is 8-bit).

Some previously accepted ways of passing search criteria will not work as they did in previous versions of IMAPClient. Small changes will be required in these cases. Here are some examples of how to update code written against older versions of IMAPClient:

```
c.search(['NOT DELETED'])    # Before
c.search(['NOT', 'DELETED']) # After

c.search(['TEXT "foo"'])     # Before
c.search(['TEXT', 'foo'])    # After (IMAPClient will add the quotes)

c.search(['DELETED', 'TEXT "foo"'])    # Before
c.search(['DELETED', 'TEXT', 'foo'])   # After

c.search(['SMALLER 1000'])    # Before
c.search(['SMALLER', 1000])   # After
```

It is also possible to pass a single string as the search criteria. IMAPClient will not attempt quoting in this case, allowing the caller to specify search criteria at a lower level. Specifying criteria using a sequence of strings is preferable however. The following examples (equivalent to those further above) are valid:

```
c.search('DELETED')
c.search('NOT DELETED')
c.search('FLAGGED SUBJECT "foo" BODY "hello world"')
c.search('NOT DELETED SMALLER 1000')
c.search('SINCE 03-May-2006')
```

Search charset

The way that the search *charset* argument is handled has also changed.

Any unicode criteria arguments will now be encoded by IMAPClient using the supplied charset. The charset must refer to an encoding that is capable of handling the criteria's characters or an error will occur. The charset must obviously also be one that the server supports! (UTF-8 is common)

Any criteria given as bytes will not be changed by IMAPClient, but the provided charset will still be passed to the IMAP server. This allows already encoding criteria to be passed through as-is. The encoding referred to by *charset* should match the actual encoding used for the criteria.

The following are valid examples:

```
c.search(['TEXT', u'\u263a'], 'utf-8')        # IMAPClient will apply UTF-8 encoding
c.search([b'TEXT', b'\xe2\x98\xba'], 'utf-8')  # Caller has already applied UTF-8
→encoding
```

The documentation and tests for search, gmail_search, sort and thread has updated to account for these changes and have also been generally improved.

8.6.4 Socket timeout support [NEW]

IMAPClient now accepts a timeout at creation time. The timeout applies while establishing the connection and for all operations on the socket connected to the IMAP server.

8.6.5 Semantic versioning

In order to better indicate version compatibility to users, IMAPClient will now strictly adhere to the Semantic Versioning scheme.

8.6.6 Performance optimisation for parsing message id lists

A short circuit is now used when parsing a list of message ids which greatly speeds up parsing time.

8.6.7 Other

- Perform quoting of Gmail labels. Thanks to Pawel Sz for the fix.
- The type of the various flag constants was fixed. Thanks to Thomi Richards for pointing this out.
- Now using mock 1.3.0. Thanks to Thomi Richards for the patch.

- Fixed handling of very long numeric only folder names. Thanks to Paweł Gorzelany for the patch.

- The default charset for gmail_search is now UTF-8. This makes it easier to use any unicode string as a search string and is safe because Gmail supports UTF-8 search criteria.

- PEP8 compliance fixed (except for some occasional long lines)

- Added a "shutdown" method.

- The embedded six package has been removed in favour of using an externally installed instance.

- Fixed handling of literals in STATUS responses.

- Only use the untagged post-login CAPABILITY response once (if sent by server).

- Release history made part of the main documentation.

- Clarified how message ids work in the docs.

- Livetest infrastructure now works with Yahoo's OAUTH2

- Fixed bytes handling in Address.__str__

8.7 Version 0.13

8.7.1 Added support for the ID command [NEW]

As per RFC2971. Thanks to Eben Freeman from Nylas.

8.7.2 Fix exception with NIL address in envelope address list

Thanks to Thomas Steinacher for this fix.

8.7.3 Fixed handling of NIL in SEARCH response

Fixed a regression in the handling of NIL/None SEARCH responses. Thanks again to Thomas Steinacher.

8.7.4 Date parsing fixes

Don't traceback when an unparsable date is seen in ENVELOPE responses. None is returned instead.

Support quirky timestamp strings which use dots for the time separator.

Removed horrible INTERNALDATE parsing code (use parse_to_datetime instead).

datetime_to_imap has been moved to the datetime_util module and is now called datetime_to_INTERNALDATE. This will only affect you in the unlikely case that you were importing this function out of the IMAPClient package.

8.7.5 Other

- The docs for various IMAPClient methods, and the HACKING.rst file have been updated.

- CONDSTORE live test is now more reliable (especially when running against Gmail)

8.8 Version 0.12

8.8.1 Fixed unicode handling [API CHANGE]

During the work to support Python 3, IMAPClient was changed to do return unicode for most responses. This was a bad decision, especially because it effectively breaks content that uses multiple encodings (e.g. RFC822 responses). This release includes major changes so that most responses are returned as bytes (Python 3) or str (Python 2). This means that correct handling of response data is now possible by code using IMAPClient.

Folder name handling has also been cleaned up as part of this work. If the `folder_encode` attribute is `True` (the default) then folder names will **always** be returned as unicode. If `folder_encode` is False then folder names will always be returned as bytes/strs.

Code using IMAPClient will most likely need to be updated to account these unicode handling changes.

Many thanks to Inbox (now Nilas, https://nilas.com/) for sponsoring this work.

8.8.2 Extra __init__ keyword args are passed through [NEW]

Any unused keyword arguments passed to the IMAPClient initialiser will now be passed through to the underlying imaplib IMAP4, IMAP4_SSL or IMAP4_stream class. This is specifically to allow the use of imaplib features that control certificate validation (if available with the version of Python being used).

Thanks to Chris Arndt for this change.

8.8.3 MODSEQ parts in SEARCH responses are now handled

If the CONDSTORE extension is supported by a server and a MODSEQ criteria was used with search(), a TypeError could occur. This has now been fixed and the MODSEQ value returned by the server is now available via an attribute on the returned list of ids.

8.8.4 Minor Changes

- Small tweaks to support Python 3.4.
- The deprecated get_folder_delimiter() method has been removed.
- More control over OAUTH2 parameters. Thanks to Phil Peterson for this.
- Fixed livetest/interact OAUTH handling under Python 3.

8.9 Version 0.11.1

- Close folders during livetest cleanup so that livetests work with newer Dovecot servers (#131)

8.10 Version 0.11

8.10.1 Support for raw Gmail searching [NEW]

The new gmail_search methods allows direct Gmail queries using the X-GM-RAW search extension. Thanks to John Louis del Rosario for the patch.

8.10.2 ENVELOPE FETCH response parsing [NEW, API CHANGE]

ENVELOPE FETCH responses are now returned as Envelope instances. These objects are namedtuples providing convenient attribute and positional based access to envelope fields. The Date field is also now converted to a datetime instance.

As part of this change various date and time related utilities were moved to a new module at imapclient.datetime_util.

Thanks to Naveen Nathan for the work on this feature.

8.10.3 Correct nested BODYSTRUCTURE handling [API CHANGE]

BODY and BODYSTRUCTURE responses are now processed recusively so multipart sections within other multipart sections are returned correctly. This also means that each part of the response now has a is_multipart property available.

NOTE: code that expects the old (broken) behaviour will need to be updated.

Thanks to Brandon Rhodes for the bug report.

8.10.4 SELECT response bug fix

Handle square brackets in flags returned in SELECT response. Previously these would cause parsing errors. Thanks to Benjamin Morrise for the bug report.

8.10.5 Minor Changes

Copyright date update for 2014.

8.11 Version 0.10.2

Switch back to setuptools now that distribute and setuptools have merged back. Some users were reporting problems with distribute and the newer versions of setuptools.

8.12 Version 0.10.1

Fixed regressions in several cases when binary data (i.e. normal strings under Python 2) are used as arguments to some methods. Also refactored input normalisation functions somewhat.

Fixed buggy method for extracting flags and Gmail labels from STORE responses.

8.13 Version 0.10

8.13.1 Python 3 support (#22) [API CHANGE]

Python 3.2 and 3.3 are now officially supported. This release also means that Python versions older than 2.6 are no longer supported.

A single source approach has been used, with no conversion step required.

A big thank you to Mathieu Agopian for his massive contribution to getting the Python 3 port finished. His changes and ideas feature heavily in this release.

IMPORTANT: Under Python 2, all strings returned by IMAPClient are now returned as unicode objects. With the exception of folder names, these unicode objects will only contain characters in the ASCII range so this shouldn't break existing code, however there is always a chance that there will be a problem. Please test your existing applications thoroughly with this verison of IMAPClient before deploying to production situations.

8.13.2 Minor Changes

- "python setup.py test" now runs the unit tests
- Mock library is now longer included (listed as external test dependency)
- live tests that aren't UID related are now only run once
- live tests now perform far less logins to the server under test
- Unit tests can now be run for all supported Python versions using `tox`.
- Improved documentation regarding working on the project.
- Many documentation fixes and improvements.

8.13.3 Minor Bug Fixes

- HIGHESTMODSEQ in SELECT response is now parsed correctly
- Fixed daylight saving handling in FixedOffset class
- Fixed –port command line bug in imapclient.interact when SSL connections are made.

8.14 Version 0.9.2

8.14.1 THREAD support [NEW]

The IMAP THREAD command is now supported. Thanks to Lukasz Mierzwa for the patches.

8.14.2 Enhanced capability querying [NEW]

Previously only the pre-authentication server capabilities were returned by the capabilities() method. Now, if the connection is authenticated, the post-authentication capabilities will be returned. If the server sent an untagged CA-PABILITY response after authentication, that will be used, avoiding an unnecessary CAPABILITY command call.

All this ensures that the client sees all available server capabilities.

8.14.3 Minor Features

- Better documentation for contributers (see HACKING file)
- Copyright date update for 2013.

8.15 Version 0.9.1

8.15.1 Stream support [NEW]

It is now possible to have IMAPClient run an external command to establish a connection to the IMAP server via a new *stream* keyword argument to the initialiser. This is useful for exotic connection or authentication setups. The *host* argument is used as the command to run.

Thanks to Dave Eckhardt for the original patch.

8.15.2 OAUTH2 Support [NEW]

OAUTH2 authentication (as supported by Gmail's IMAP) is now available via the new oauth2_login method. Thanks to Zac Witte for the original patch.

8.15.3 livetest now handles Gmail's new message handling

Gmail's IMAP implementation recently started requiring a NOOP command before new messages become visible after delivery or an APPEND. The livetest suite has been updated to deal with this.

8.16 Version 0.9

8.16.1 Gmail Label Support

New methods have been added for interacting with Gmail's label API: get_gmail_labels, add_gmail_labels, set_gmail_labels, remove_gmail_labels. Thanks to Brian Neal for the patches.

8.16.2 Removed Code Duplication (#9)

A signficant amount of duplicated code has been removed by abstracting out common command handling code. This will make the Python 3 port and future maintenance easier.

8.16.3 livetest can now be run against non-dummy accounts (#108)

Up until this release the tests in imapclient.livetest could only be run against a dummy IMAP account (all data in the account would be lost during testing). The tests are now limited to a sub-folder created by the tests so it is ok to run them against an account that contains real messages. These messages will be left alone.

8.16.4 Minor Features

- Don't traceback when an IMAP server returns a all-digit folder name without quotes. Thanks to Rhett Garber for the bug report. (#107)
- More tests for ACL related methods (#89)
- More tests for namespace()
- Added test for read-only select_folder()

8.16.5 Minor Bug Fixes

- Fixed rename live test so that it uses folder namespaces (#100).

- Parse STATUS responses robustly - fixes folder_status() with MS Exchange.

- Numerous livetest fixes to work around oddities with the MS Exchange IMAP implementation.

8.17 Version 0.8.1

- IMAPClient wasn't installing on Windows due to an extra trailing slash in MANIFEST.in (#102). This is a bug in distutils.

- MANIFEST.in was fixed so that the main documentation index file is included the source distribution.

- distribute_setup.py was updated to the 0.6.24 version.

- This release also contains some small documentation fixes.

8.18 Version 0.8

8.18.1 OAUTH Support (#54) [NEW]

OAUTH authentication is now supported using the oauth_login method. This requires the 3rd party oauth2 package is installed. Thanks to Johannes Heckel for contributing the patch to this.

8.18.2 IDLE Support (#50) [NEW]

The IDLE extension is now supported through the new idle(), idle_check() and idle_done() methods. See the example in imapclient/examples/idle_example.py.

8.18.3 NOOP Support (#74) [NEW]

The NOOP command is now supported. It returns parsed untagged server responses in the same format as idle_check() and idle_done().

8.18.4 Sphinx Based Docs (#5) [NEW]

Full documentation is now available under doc/html in the source distribution and at http://imapclient.readthedocs.io/ online.

8.18.5 Added rename_folder (#77) [NEW]

Renaming of folders was an obvious omission!

8.18.6 Minor Features

- interact.py can now read livetest.py INI files (#66)

- interact.py can now embed shells from ipython 0.10 and 0.11 (#98)

- interact.py and livetest.py are now inside the imapclient package so they can be used even when IMAClient has been installed from PyPI (#82)

- Added "debug" propety and setting of a log file (#90)

- "normalise_times" attribute allows caller to select whether datetimes returned by fetch() are native or not (#96) (Thanks Andrew Scheller)

- Added imapclient.version_info - a tuple that contains the IMAPClient version number broken down into it's parts.

8.18.7 Minor Bug Fixes

- getacl() was using wrong lexing class (#85) (Thanks josephhh)

- Removed special handling for response tuples without whitespace between them. Post-process BODY/BODYSTRUCTURE responses instead. This should not affect the external API. (#91) (Thanks daishi)

- Fix incorrect msg_id for UID fetch when use_uid is False (#99)

8.19 Version 0.7

8.19.1 BODY and BODYSTRUCTURE parsing fixes (#58) [API CHANGE]

The response values for BODY and BODYSTRUCTURE responses may include a sequence of tuples which are not separated by whitespace. These should be treated as a single item (a list of multiple arbitrarily nested tuples) but IMAPClient was treating them as separate items. IMAPClient now returns these tuples in a list to allow for consistent parsing.

A BODYSTRUCTURE response for a multipart email with 2 parts would have previously looked something like this:

```
(('text', 'html', ('charset', 'us-ascii'), None, None, 'quoted-printable', 55, 3),
 ('text', 'plain', ('charset', 'us-ascii'), None, None, '7bit', 26, 1),
 'mixed', ('boundary', '================1534046211=='))
```

The response is now returned like this:

```
([
    ('text', 'html', ('charset', 'us-ascii'), None, None, 'quoted-printable', 55, 3),
    ('text', 'plain', ('charset', 'us-ascii'), None, None, '7bit', 26, 1)
 ],
 'mixed', ('boundary', '================1534046211=='))
```

The behaviour for single part messages is unchanged. In this case the first element of the tuple is a string specifying the major content type of the message (eg "text").

An is_multipart boolean property now exists on BODY and BODYSTRUCTURE responses to allow the caller to easily determine whether the response is for a multipart message.

Code that expects the previous response handling behaviour needs to be updated.

8.19.2 Live tests converted to use unittest2 (#4)

livetest.py now uses the unittest2 package to run the tests. This provides much more flexibility that the custom approach that was used before. Dependencies between tests are gone - each test uses a fresh IMAP connection and is preceeded by the same setup.

unittest2.main() is used to provide a number of useful command line options and the ability to run a subset of tests.

IMAP account parameters are now read using a configuration file instead of command line arguments. See livetest-sample.ini for an example.

8.19.3 Added NAMESPACE support (#63) [API CHANGE]

namespace() method added and get_folder_delimiter() has been deprecated.

8.19.4 Added support for FETCH modifiers (#62) [NEW]

The fetch method now takes optional modifiers as the last argument. These are required for extensions such as RFC 4551 (conditional store). Thanks to Thomas Jost for the patch.

8.20 Version 0.6.2

8.20.1 Square brackets in responses now parsed correctly (#55)

This fixes response handling for FETCH items such as `BODY[HEADER.FIELDS (from subject)]`.

8.20.2 Example moved (#56)

The example has been moved to imapclient/examples directory and is included when the IMAPClient is installed from PyPI.

8.20.3 Distribute (#57)

The project is now packaged using Distribute instead of setuptools. There should be no real functional change.

8.21 Version 0.6.1

8.21.1 Python SSL bug patch

Automatically patch a bug in imaplib which can cause hangs when using SSL (Python Issue 5949). The patch is only applied when the running Python version is known to be affected by the problem.

8.21.2 Doc update

Updated the README to better reflect the current state of the project.

8.22 Version 0.6

8.22.1 New response parser (#1, #45)

Command response lexing and parsing code rewritten from stratch to deal with various bugs that surfaced when dealing with more complex responses (eg. BODYSTRUCTURE and ENVELOPE). This change also fixes various problems when interacting with Gmail and MS Exchange.

8.22.2 XLIST extension support (#25) [NEW]

Where the server supports it, xlist_folders() will return a mapping of various common folder names to the actual server folder names. Gmail's IMAP server supports this.

8.22.3 Added COPY command support (#36) [NEW]

New copy() method.

8.22.4 Added interact.py [NEW]

A script for interactive IMAPClient sessions. Useful for debugging and exploration. Uses IPython if installed.

8.22.5 Full SELECT response (#24) [API CHANGE]

select_folder() now returns a dictionary with the full (parsed) SELECT command response instead of just the message count.

8.22.6 Full list responses (#24) [API CHANGE]

The return value from list_folders(), list_sub_folders() and xlist_folders() now include the IMAP folder flags and delimiter.

8.22.7 Folder name character encoding (#21) [API CHANGE]

Bytes that are greater than 0x7f in folder names are will cause an exception when passed to methods that accept folder name arguments because there is no unambigous way to handle these. Callers should encode such folder names to unicode objects first.

Folder names are now always returned as unicode objects.

8.22.8 Message sequence number now always returned in FETCH responses

Fetch responses now include a "SEQ" element which gives the message (non-UID) sequence number. This allows for easy mapping between UIDs and standard sequence IDs.

8.22.9 Folder name handling fixes (#28, #42)

Various folder name handling bugs fixed.

8.23 Version 0.5.2

8.23.1 Folder name quoting and escaping fixes (#28)

Correctly handle double quotes and backslashes in folder names when parsing LIST and LSUB responses.

8.23.2 Fixed fetch literal handling (#33)

Fixed problem with parsing responses where a literal followed another literal.

8.24 Version 0.5.1

8.24.1 License change

Changed license from GPL to new BSD.

8.25 Version 0.5

8.25.1 SSL support

Support for SSL based connections by passing ssl=True when constructing an IMAPClient instance.

8.25.2 Transparent folder encoding

Folder names are now encoded and decoded transparently if required (using modified UTF-7). This means that any methods that return folder names may return unicode objects as well as normal strings [API CHANGE]. Additionally, any method that takes a folder name now accepts unicode object too. Use the folder_encode attribute to control whether encode/decoding is performed.

8.25.3 Unquoted folder name handling fix

Unquoted folder names in server responses are now handled correctly. Thanks to Neil Martinsen-Burrell for reporting this bug.

8.25.4 Fixed handling of unusual characters in folder names

Fixed a bug with handling of unusual characters in folder names.

8.25.5 Timezone handling [API CHANGE]

Timezones are now handled correctly for datetimes passed as input and for server responses. This fixes a number of bugs with timezones. Returned datetimes are always in the client's local timezone.

8.25.6 More unit tests

Many more unit tests added, some using Michael Foord's excellent mock.py. (http://www.voidspace.org.uk/python/mock/)

Python Module Index

i

Index