smtplib — Simple Mail Transfer Protocol Client

Purpose: Interact with SMTP servers, including sending email.

`smtplib` includes the class SMTP, which can be used to communicate with mail servers to send mail.

Note

The email addresses, host names, and IP addresses in the following examples have been obscured, but otherwise the transcripts illustrate the sequence of commands and responses accurately.

Sending an Email Message

The most common use of SMTP is to connect to a mail server and send a message. The mail server host name and port can be passed to the constructor, or `connect()` can be invoked explicitly. Once connected, call `sendmail()` with the envelope parameters and body of the message. The message text should be fully formed and comply with RFC 5322, since `smtplib` does not modify the contents or headers at all. That means the `From` and `To` headers need to be added by the caller.

```
# smtplib_sendmail.py

import smtplib
import email.utils
from email.mime.text import MIMEText

# Create the message
msg = MIMEText('This is the body of the message.')
msg['To'] = email.utils.formataddr(('Recipient',
                                    'recipient@example.com'))
msg['From'] = email.utils.formataddr(('Author',
                                      'author@example.com'))
msg['Subject'] = 'Simple test message'

server = smtplib.SMTP('localhost', 1025)
server.set_debuglevel(True)  # show communication with the server
try:
    server.sendmail('author@example.com',
                    ['recipient@example.com'],
                    msg.as_string())
finally:
    server.quit()
```

In this example, debugging is also turned on to show the communication between client and server. Otherwise the example would produce no output at all.

```
$ python3 smtplib_sendmail.py

send: 'ehlo 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.ip6.arpa\r\n'
reply: b'250-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.ip6.arpa\r\n'
reply: b'250-SIZE 33554432\r\n'
reply: b'250 HELP\r\n'
reply: retcode (250); Msg: b'1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa\nSIZE 33554432\nHELP'
send: 'mail FROM:<author@example.com> size=236\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'rcpt TO:<recipient@example.com>\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'data\r\n'
reply: b'354 End data with <CR><LF>.<CR><LF>\r\n'
```

```
reply: retcode (354); Msg: b'End data with <CR><LF>.<CR><LF>'
data: (354, b'End data with <CR><LF>.<CR><LF>')
send: b'Content-Type: text/plain; charset="us-ascii"\r\nMIME-Ver
sion: 1.0\r\nContent-Transfer-Encoding: 7bit\r\nTo: Recipient <r
ecipient@example.com>\r\nFrom: Author <author@example.com>\r\nSu
bject: Simple test message\r\n\r\nThis is the body of the messag
e.\r\n.\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
data: (250, b'OK')
send: 'quit\r\n'
reply: b'221 Bye\r\n'
reply: retcode (221); Msg: b'Bye'
```

The second argument to `sendmail()`, the recipients, is passed as a list. Any number of addresses can be included in the list to have the message delivered to each of them in turn. Since the envelope information is separate from the message headers, it is possible to blind carbon-copy (BCC) someone by including them in the method argument, but not in the message header.

Authentication and Encryption

The SMTP class also handles authentication and TLS (transport layer security) encryption, when the server supports them. To determine if the server supports TLS, call `ehlo()` directly to identify the client to the server and ask it what extensions are available. Then call `has_extn()` to check the results. After TLS is started, `ehlo()` must be called again before authenticating. Many mail hosting providers now *only* support TLS-based connections. For communicating with those servers, use SMTP_SSL to start off with an encrypted connection.

```
# smtplib_authenticated.py

import smtplib
import email.utils
from email.mime.text import MIMEText
import getpass

# Prompt the user for connection info
to_email = input('Recipient: ')
servername = input('Mail server name: ')
serverport = input('Server port: ')
if serverport:
    serverport = int(serverport)
else:
    serverport = 25
use_tls = input('Use TLS? (yes/no): ').lower()
username = input('Mail username: ')
password = getpass.getpass("%s's password: " % username)

# Create the message
msg = MIMEText('Test message from PyMOTW.')
msg.set_unixfrom('author')
msg['To'] = email.utils.formataddr(('Recipient', to_email))
msg['From'] = email.utils.formataddr(('Author',
                                      'author@example.com'))
msg['Subject'] = 'Test from PyMOTW'

if use_tls == 'yes':
    print('starting with a secure connection')
    server = smtplib.SMTP_SSL(servername, serverport)
else:
    print('starting with an insecure connection')
    server = smtplib.SMTP(servername, serverport)
try:
    server.set_debuglevel(True)

    # identify ourselves, prompting server for supported features
    server.ehlo()

    # If we can encrypt this session, do it
    if server.has_extn('STARTTLS'):
        print('(starting TLS)')
        server.starttls()
        server.ehlo()  # reidentify ourselves over TLS connection
```

```
    else:
        print('(no STARTTLS)')

    if server.has_extn('AUTH'):
        print('(logging in)')
        server.login(username, password)
    else:
        print('(no AUTH)')

    server.sendmail('author@example.com',
                    [to_email],
                    msg.as_string())
finally:
    server.quit()
```

The STARTTLS extension does not appear in the reply to EHLO after TLS is enabled.

```
$ python3 source/smtplib/smtplib_authenticated.py
Recipient: doug@pymotw.com
Mail server name: localhost
Server port: 1025
Use TLS? (yes/no): no
Mail username: test
test's password:
starting with an insecure connection
send: 'ehlo 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.ip6.arpa\r\n'
reply: b'250-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.ip6.arpa\r\n'
reply: b'250-SIZE 33554432\r\n'
reply: b'250 HELP\r\n'
reply: retcode (250); Msg: b'1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa\nSIZE 33554432\nHELP'
(no STARTTLS)
(no AUTH)
send: 'mail FROM:<author@example.com> size=220\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'rcpt TO:<doug@pymotw.com>\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'data\r\n'
reply: b'354 End data with <CR><LF>.<CR><LF>\r\n'
reply: retcode (354); Msg: b'End data with <CR><LF>.<CR><LF>'
data: (354, b'End data with <CR><LF>.<CR><LF>')
send: b'Content-Type: text/plain; charset="us-ascii"\r\n
MIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\nTo:
Recipient <doug@pymotw.com>\r\nFrom: Author <author@example.com>
\r\nSubject: Test from PyMOTW\r\n\r\nTest message from PyMOTW.
\r\n.\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
data: (250, b'OK')
send: 'quit\r\n'
reply: b'221 Bye\r\n'
reply: retcode (221); Msg: b'Bye'

$ python3 source/smtplib/smtplib_authenticated.py
Recipient: doug@pymotw.com
Mail server name: mail.isp.net
Server port: 465
Use TLS? (yes/no): yes
Mail username: doughellmann@isp.net
doughellmann@isp.net's password:
starting with a secure connection
send: 'ehlo 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.ip6.arpa\r\n'
reply: b'250-mail.isp.net\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-SIZE 71000000\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
```

```
reply: b'250-8BITMIME\r\n'
reply: b'250-AUTH PLAIN LOGIN\r\n'
reply: b'250 AUTH=PLAIN LOGIN\r\n'
reply: retcode (250); Msg: b'mail.isp.net\nPIPELINING\nSIZE
71000000\nENHANCEDSTATUSCODES\n8BITMIME\nAUTH PLAIN LOGIN\n
AUTH=PLAIN LOGIN'
(no STARTTLS)
(logging in)
send: 'AUTH PLAIN AGRvdWdoZWxsbWFubkBmYXN0bWFpbC5mbQBUTUZ3MDBmZmF
zdG1haWw=\r\n'
reply: b'235 2.0.0 OK\r\n'
reply: retcode (235); Msg: b'2.0.0 OK'
send: 'mail FROM:<author@example.com> size=220\r\n'
reply: b'250 2.1.0 Ok\r\n'
reply: retcode (250); Msg: b'2.1.0 Ok'
send: 'rcpt TO:<doug@pymotw.com>\r\n'
reply: b'250 2.1.5 Ok\r\n'
reply: retcode (250); Msg: b'2.1.5 Ok'
send: 'data\r\n'
reply: b'354 End data with <CR><LF>.<CR><LF>\r\n'
reply: retcode (354); Msg: b'End data with <CR><LF>.<CR><LF>'
data: (354, b'End data with <CR><LF>.<CR><LF>')
send: b'Content-Type: text/plain; charset="us-ascii"\r\n
MIME-Version: 1.0\r\nContent-Transfer-Encoding: 7bit\r\nTo:
Recipient <doug@pymotw.com>\r\nFrom: Author <author@example.com>
\r\nSubject: Test from PyMOTW\r\n\r\nTest message from PyMOTW.
\r\n.\r\n'
reply: b'250 2.0.0 Ok: queued as A0EF7F2983\r\n'
reply: retcode (250); Msg: b'2.0.0 Ok: queued as A0EF7F2983'
data: (250, b'2.0.0 Ok: queued as A0EF7F2983')
send: 'quit\r\n'
reply: b'221 2.0.0 Bye\r\n'
reply: retcode (221); Msg: b'2.0.0 Bye'
```

Verifying an Email Address

The SMTP protocol includes a command to ask a server whether an address is valid. Usually VRFY is disabled to prevent spammers from finding legitimate email addresses, but if it is enabled a client can ask the server about an address and receive a status code indicating validity along with the user's full name, if it is available.

```
# smtplib_verify.py
```

```
import smtplib

server = smtplib.SMTP('mail')
server.set_debuglevel(True)  # show communication with the server
try:
    dhellmann_result = server.verify('dhellmann')
    notthere_result = server.verify('notthere')
finally:
    server.quit()

print('dhellmann:', dhellmann_result)
print('notthere :', notthere_result)
```

As the last two lines of output here show, the address `dhellmann` is valid but `notthere` is not.

```
$ python3 smtplib_verify.py

send: 'vrfy <dhellmann>\r\n'
reply: '250 2.1.5 Doug Hellmann <dhellmann@mail>\r\n'
reply: retcode (250); Msg: 2.1.5 Doug Hellmann <dhellmann@mail>
send: 'vrfy <notthere>\r\n'
reply: '550 5.1.1 <notthere>... User unknown\r\n'
reply: retcode (550); Msg: 5.1.1 <notthere>... User unknown
send: 'quit\r\n'
reply: '221 2.0.0 mail closing connection\r\n'
reply: retcode (221); Msg: 2.0.0 mail closing connection
dhellmann: (250, '2.1.5 Doug Hellmann <dhellmann@mail>')
notthere : (550, '5.1.1 <notthere>... User unknown')
```

```
nothere : (550, '5.1.1 <nothere>... User unknown')
```

See also

- Standard library documentation for smtplib
- **RFC 821** – The Simple Mail Transfer Protocol (SMTP) specification.
- **RFC 1869** – SMTP Service Extensions to the base protocol.
- **RFC 822** – “Standard for the Format of ARPA Internet Text Messages”, the original email message format specification.
- **RFC 5322** – “Internet Message Format”, updates to the email message format.
- `email` – Standard library module for building and parsing email messages.
- `smtpd` – Implements a simple SMTP server.

Email

smtpd — Sample Mail Servers

Quick Links

Sending an Email Message
Authentication and Encryption
Verifying an Email Address

This page was last updated 2016-12-18.

Navigation

Email
smtpd — Sample Mail Servers

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

Other Writing

Blog
The Python Standard Library By Example