smtpd — Sample Mail Servers

Purpose: Includes classes for implementing SMTP servers.

The `smtpd` module includes classes for building simple mail transport protocol servers. It is the server-side of the protocol used by `smtplib`.

Mail Server Base Class

The base class for all of the provided example servers is `SMTPServer`. It handles communicating with the client, receiving incoming data, and provides a convenient hook to override to process the message once it is fully available.

The constructor arguments are the local address to listen for connections and the remote address where proxied messages should be delivered. The method `process_message()` is provided as a hook to be overridden by a derived class. It is called when the message is completely received, and given these arguments:

`peer`

The client's address, a tuple containing IP and incoming port.

`mailfrom`

The "from" information out of the message envelope, given to the server by the client when the message is delivered. This does not necessarily match the `From` header in all cases.

`rcpttos`

The list of recipients from the message envelope. Again, this does not always match the `To` header, especially if a recipient is being blind carbon copied.

`data`

The full RFC 5322 message body.

The default implementation of `process_message()` raises `NotImplementedError`. The next example defines a subclass that overrides the method to print information about the messages it receives.

```
# smtpd_custom.py

import smtpd
import asyncore


class CustomSMTPServer(smtpd.SMTPServer):

    def process_message(self, peer, mailfrom, rcpttos, data):
        print('Receiving message from:', peer)
        print('Message addressed from:', mailfrom)
        print('Message addressed to  :', rcpttos)
        print('Message length        :', len(data))


server = CustomSMTPServer(('127.0.0.1', 1025), None)

asyncore.loop()
```

`SMTPServer` uses `asyncore`, so to run the server call `asyncore.loop()`.

A client is needed to demonstrate the server. One of the examples from the section on `smtplib` can be adapted to create a client to send data to the test server running locally on port 1025.

```
# smtpd_senddata.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('127.0.0.1', 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()
```

To test the programs, run `smtpd_custom.py` in one terminal and `smtpd_senddata.py` in another.

```
$ python3 smtpd_custom.py

Receiving message from: ('127.0.0.1', 58541)
Message addressed from: author@example.com
Message addressed to  : ['recipient@example.com']
Message length        : 229
```

The debug output from `smtpd_senddata.py` shows all of the communication with the server.

```
$ python3 smtpd_senddata.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
.0.0.0.0.0.0.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
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.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'
```

To stop the server, press `Ctrl-C`.

Debugging Server

The previous example shows the arguments to `process_message()`, but `smtpd` also includes a server specifically designed for more complete debugging, called `DebuggingServer`. It prints the entire incoming message to the console and then stops processing (it does not proxy the message to a real mail server).

```
# smtpd_debug.py

import smtpd
import asyncore

server = smtpd.DebuggingServer(('127.0.0.1', 1025), None)

asyncore.loop()
```

Using the `smtpd_senddata.py` client program from earlier, the output of the `DebuggingServer` is:

```
---------- MESSAGE FOLLOWS ----------
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
To: Recipient <recipient@example.com>
From: Author <author@example.com>
Subject: Simple test message
X-Peer: 127.0.0.1

This is the body of the message.
------------ END MESSAGE ------------
```

Proxy Server

The `PureProxy` class implements a straightforward proxy server. Incoming messages are forwarded upstream to the server given as argument to the constructor.

Warning

The standard library documentation for `smtpd` says, “running this has a good chance to make you into an open relay, so please be careful.”

The steps for setting up the proxy server are similar to the debug server.

```
# smtpd_proxy.py

import smtpd
import asyncore

server = smtpd.PureProxy(('127.0.0.1', 1025), ('mail', 25))

asyncore.loop()
```

It prints no output, though, so to verify that it is working look at the mail server logs.

```
Aug 20 19:16:34 homer sendmail[6785]: m9JNGXJb006785:
from=<author@example.com>, size=248, class=0, nrcpts=1,
msgid=<200810192316.m9JNGXJb006785@homer.example.com>,
proto=ESMTP, daemon=MTA, relay=[192.168.1.17]
```

See also

- Standard library documentation for smtpd
- `smtplib` – Provides a client interface.
- `email` – Parses email messages.
- `asyncore` – Base module for writing asynchronous servers.
- **RFC 2822** – *Internet Message Format*, defines the email message format.
- **RFC 5322** – Replacement for RFC 2822.

Quick Links
Mail Server Base Class
Debugging Server
Proxy Server

This page was last updated 2016-12-18.

Navigation
smtplib — Simple Mail Transfer Protocol Client
mailbox — Manipulate Email Archives

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